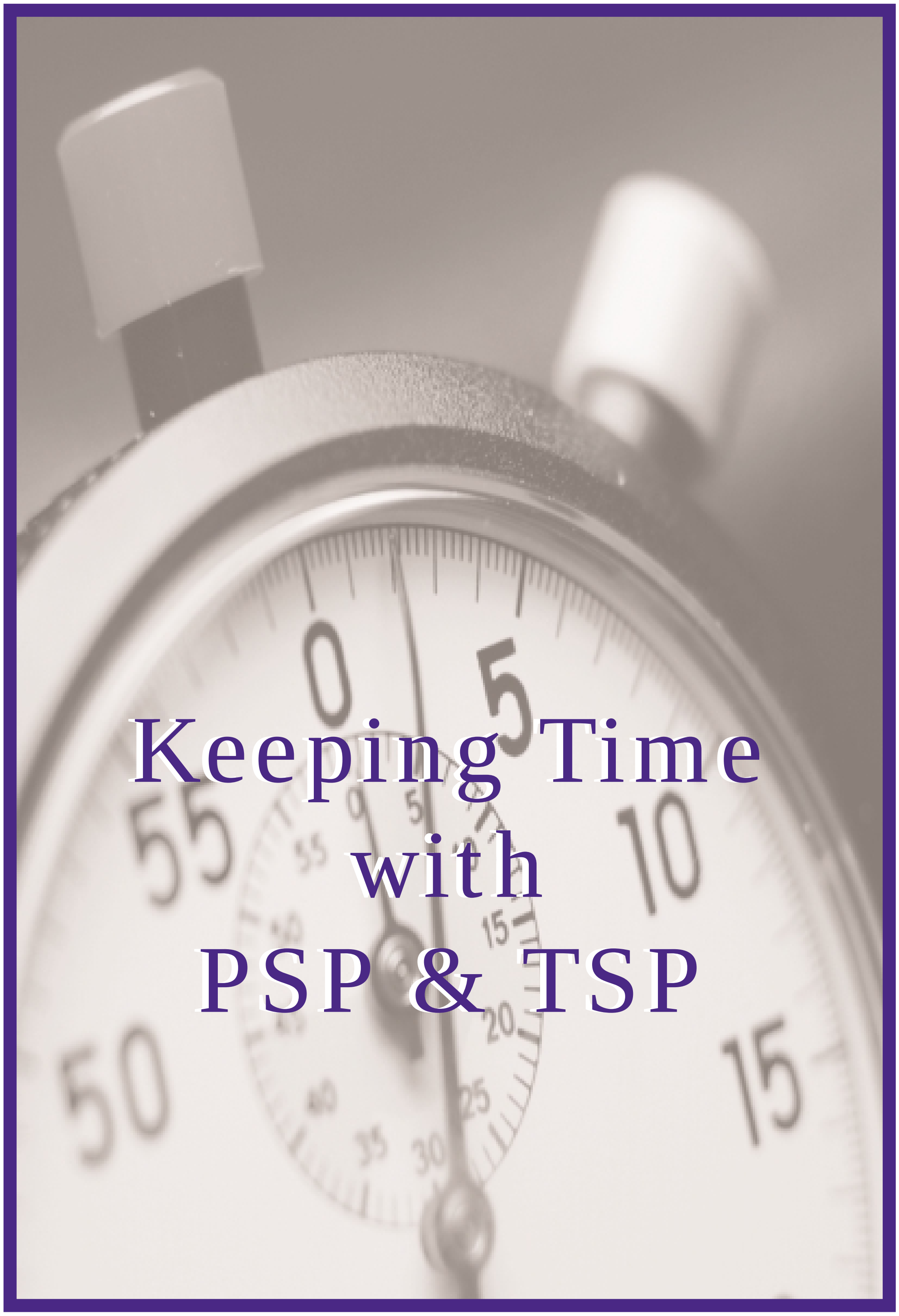
CrossTalk

June 2000 *The Journal of Defense Software Engineering* Vol. 13 No. 6



Keeping Time with PSP & TSP

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **JUN 2000**	2. REPORT TYPE	3. DATES COVERED **00-00-2000 to 00-00-2000**
4. TITLE AND SUBTITLE **CrossTalk. The Journal of Defense Software Engineering. Volume 13, Number 6, June 2000**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave,Hill AFB,UT,84056-5820**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**37**	

PSP/ TSP

4 **Building Productive Teams**
It is not practical for one person to do an entire systems engineering job; hence, we need to build productive teams.
by Watts S. Humphrey

7 **Managing Risk with TSP**
The Team Software Process regarding risk has come in handy at Hill Air Force Base.
by David R. Webb

11 **Making Quality Happen: The Managers' Role**
An Advanced Information Systems case study based on two projects in Chennai, India.
by Girish Seshagiri

14 **PSP: Fair Warning**
The Personal Software Process Course is difficult and time consuming. Students beware.
by Elizabeth Starrett

Software Engineering Technology

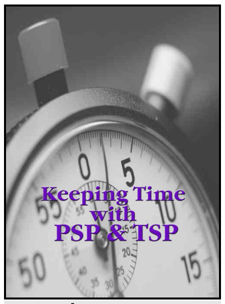
16 **Statistical Process Control Meets Earned Value**
Presenting an approach for software production management, i.e., cost and schedule control.
by Walt Lipke and Jeff Vaughn

21 **What We Have Learned**
From the four SEI core metrics a fifth is derived—productivity. All five are interrelated.
by Lawrence H. Putnam and Ware Myers

25 **Proven Techniques for Efficiently Generating and Testing Software**
This paper presents a proven process that uses advanced tools to design, develop and test optimal software.
by Keith R. Wegner

Open Forum

29 **Large Software Systems–Back to Basics**
Development methods that work on small problems seem to not scale well to larger, more complicated problems.
by John R. Evans



On the Cover:
Brandon Scott is a graduate of Utah Career College, where he studied the latest multimedia techniques and software programs. He enjoys everything from abstract design to Web design.

Departments

3 From the Publisher
10 Coming Events
13 PSP/TSP Web Sites
24 Web Addition
29 Letter to the Editor
31 Call for Articles
31 Subscription Request Form
31 BackTalk

CrossTalk

Sponsor	H. Bruce Allgood
Publisher	Reuel S. Alder
Associate Publisher	Lynn Silver
Managing Editor	Kathy Gurchiek
Associate Editor/Layout	Matthew Welker
Associate Editor/Features	Heather Winward
Voice	801-775-5555
Fax	801-777-8069
E-mail	crosstalk.staff@hill.af.mil
STSC Online	http://www.stsc.hill.af.mil
CrossTalk Online	http://www.stsc.hill.af.mil/Crosstalk/crostalk.html
CRSIP Online	http://www.crsip.hill.af.mil

Subscriptions : Send correspondence concerning subscriptions and changes of address to the following address. You may use the form on page 31.
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, Utah 84056-5205

Article Submissions : We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Guidelines for CrossTalk Authors, available upon request. We do not pay for submissions. Articles published in CrossTalk remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with CrossTalk.

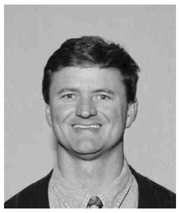
Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

Coming Events : We often list conferences, seminars, symposiums, etc., that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CrossTalk Editorial Department.
STSC Online Services: at http://www.stsc.hill.af.mil.
Call 801-777-7026, e-mail randy.schreifels@hill.af.mil.

Back Issues Available: The STSC sometimes has extra copies of back issues of CrossTalk available free of charge.

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

PSP & TSP–*The Necessary Approach*



In the fall of 1996, approximately 20 software engineers from the Software Engineering Division at Hill Air Force Base were trained in the "Personal Software Process (PSP^{SM})." I was one of them. The division of approximately 500 scientists and engineers decided to try a pilot course to see if there was value in PSP training. The course consists of a very structured and disciplined approach for writing software. Ten computer programs and five reports are assigned. The objective is to track time, lines of code used, defects inserted and removed, and the process block where it occurred. PSP techniques reduce the number of defects inserted into the code by using process reviews at early stages of computer programming and closing the gap between estimates and actuals.

"It is better to eliminate the insertion of defects instead of using the compiler to catch problems in the code." Sounds easy, right? This course is anything but easy. Among the 20 engineers in the course, most had undergraduate and master's degrees in electrical engineering or computer science. Most were leads of projects that required an enormous amount of time. Finally, most of these engineers were helping define the processes used and needed for the division's work into CMM Level 5. Those 20 agreed that it was the hardest course they had ever taken. The PSP course consumed all other responsibilities and time. Elizabeth Starrett's article, *PSP: Fair Warning*, on page 14 explains some of the difficulties in learning PSP.

After the grueling few weeks of the course, few were able to adopt the techniques taught in PSP and most did not. I was able to use many of the concepts in my Software Quality Assurance team. The PSP concepts were crucial for understanding CMM® Level 5 principles in helping the division. Those who were not able to use PSP in their projects had a harder time understanding true defect prevention and true teamwork.

Since the pilot, the division has trained many more engineers in PSP. The mission planning software project, TaskView, has brought PSP-trained engineers together into a special team, where the Team Software Process (TSP^{SM}) is being used. In *Managing Risk with the Team Software Process* on page 7, David Webb, the Technical Program Manager of TaskView, explains how TSP is an effective method of managing software project risks by using a common-sense approach with nearly defect-free code.

Great teams, whether in sports or in business, share common commitments and goals. Teamwork is essential for most successful software engineering projects, as explained in Watts Humphrey's article, *Building Productive Teams,* on page 4.

Being associated with a CMM® Level 5 organization, I believe that PSP and TSP are essential for understanding true Level 5 concepts. If your organization is striving for CMM® Level 5, it should look into investing time, money, and effort into training key players in your organization. At the very least, SEPG and SQA members should be trained in PSP and TSP.

[illegible]

Lynn P. Silver
Associate Publisher

^{SM} The Team Software Process and TSP are service marks of Carnegie Mellon University.
® The Capability and Maturity Model and CMM are Registered in the U.S. Patent and Trademark Office to Carnegie Mellon University.
^{SM} The Personal Software Process and PSP are service marks of Carnegie Mellon University.

Building Productive Teams

by Watts S. Humphrey
Software Engineering Institute, Carnegie Mellon University

The Software Engineering Institute's (SEI) Team Software Process (TSP^SM^) is a framework and a process structure for building and guiding integrated engineering teams, which are essential in development of today's complex, increasingly sophisticated systems. This paper discusses characteristics of productive teams, how to build/launch such a team, and team member preparation.

Teams are required for most engineering projects. While some small hardware or software products can be developed by individuals, the scale and complexity of modern systems is such, and the demand for short schedules so great, that it is not practical for one person to do the entire job. Systems development is a team activity, and the effectiveness of the team largely determines the quality of the engineering.

Modern systems are becoming increasingly sophisticated. Aircraft, automobiles, computer printers, television sets, and even electric razors contain software, often lots of software, and the amounts of software have been rapidly increasing. The design of such systems is vastly more complex than it was a few years ago. While there are still many modest-sized systems, the trend is for the software content of just about every product to increase by 10 or more times every 10 years. This trend has been more or less followed for several decades, and it appears likely to continue for the foreseeable future.

While most large systems involve many technologies, it is generally the software that integrates all the pieces into a cohesive whole. The software engineers provide the glue that holds the system together. It is critical that the software professionals use disciplined methods. If they do not, the integration job and the software glue that binds the system will likely have quality problems.

While every technology is important, if the software people do not properly plan and manage their work, the project will almost certainly get into trouble. That is the reason that the Software Engineering Institute (SEI) developed a framework and a process structure for building and guiding integrated engineering teams. It is called the Team Software Process (TSP^{SM}).

The TSP is one of a family of methods that can help engineering teams more effectively develop and support software-intensive systems. The Capability Maturity Model (CMM®) provides the overall improvement framework needed for effective engineering work [1]. The Personal Software Process (PSP^{SM}) provides the engineering disciplines engineers need to consistently use a defined, planned, and measured process [2]. The TSP couples the principles of integrated product teams with the PSP and CMM® methods to produce highly productive teams.

A team is more than just a group of people who happen to work together. Teamwork takes practice and involves special skills. Teams require common processes; they need agreed-upon goals; and they need effective guidance and leadership. The methods for guiding and leading such teams are well known, but they are not obvious. The SEI has developed the TSP to guide engineers and their managers in using effective teamwork methods.

This paper describes the principles behind the TSP's development. Starting with an overview of the characteristics of productive teams, the paper describes how organizations can build teams that have these characteristics. Next, the paper describes the ways the TSP process guides team formation. It closes with a summary of the preparation required for TSP team members.

The Characteristics of Productive Teams

There are different kinds of teams. In sports, for example, a basketball team's positions are dynamic while baseball team members have more static roles. In both cases, however, the members must all work together cooperatively. Conversely, wrestling and track teams are composed of individual competitors who, while not dynamically interacting, support each other socially and emotionally.

In engineering, development teams often behave much like baseball or basketball teams. While they may have multiple specialties, all the members work toward a single objective. On systems maintenance and enhancement teams, however, the engineers often work relatively independently, much like wrestling and track teams. However, regardless of the team type, productive engineering teams have certain common characteristics.

A team is a group of people who share a common goal. They must all be committed to this goal and have a common working framework. The following definition for a team has been adapted from Jean L. Dyer [3]:

- A team consists of at least two people.
- They work toward a common goal.
- Each person has been assigned specific roles.
- Completion of the mission requires some form of dependency among the group members.

Conditions for Effective Teamwork

The four parts of this definition of a team are all important. For example, it is obvious that a team must have more than one member, and the need for common goals is also generally accepted. It is not as obvious, however, why team members must have roles. Roles are essential because they provide a sense of ownership and belonging. They help guide team members on how to do their jobs; they prevent conflicts, duplicate work, and wasted effort; and they provide the members a degree of control over their working environment. Such a sense of control is a fundamental requirement for motivated and energetic team members.

Interdependence is also important. This is where each team member depends to some degree on the performance of the other members. Interdependence improves individual performance because, with complementary skills, the members can help and support each other. For example, design teams generally produce better designs than any individual member could have produced alone. Team performance is further enhanced by the social support of membership. Human beings are social animals and few people like to work entirely by themselves, at least not for very long. Because of this social context, the members will make a special effort to meet their obligations to the rest of the team.

Through mutual support and interdependence, teams become more than just the sum of their members. As teams build a trusting and cohesive relationship, they develop a spirit and an energy that can produce extraordinary results.

Innovative Teams

Another characteristic of productive teams is their ability to innovate. Innovation has been essential in the development of modern society. Innovation is more than just thinking up bright ideas, it requires creativity and a lot of hard work. Just about every engineering task is part of an innovative endeavor. This is true for the development, enhancement, and repair of complex systems. These innovative teams must produce quality products while using unfamiliar and often unproven tools and technologies. They often start projects with only partially defined needs and they must be sensitive to the user's evolving requirements.

Innovative teams must have skilled and capable people who are highly motivated. They must be creative, flexible, and disciplined. They must strive to meet demanding schedules while adjusting to changing user needs. They must also control costs and schedules while keeping management informed of their progress. In short, innovative teams have a great deal to do.

A Trusting Environment

To be creative and productive, engineering teams must work in a trusting and supportive environment [4]. Engineering teams are composed of extremely capable people who can quickly sense a lack of trust. When managers do not trust their teams to make aggressive schedules or to strive to meet these schedules, the engineers will know it. When engineers do not feel trusted and respected, they will feel antagonized and manipulated. They will no longer feel loyal to the organization and can easily lose their commitment to the team.

Since people are generally more productive when faced with an important and meaningful challenge, it is appropriate for management to challenge their teams with aggressive goals. But when the teams respond to the challenge with a plan, management must be willing to negotiate realistic commitments the engineers believe they can meet. Few people will work diligently to meet a seemingly hopeless project schedule.

Producing Productive Teams

In summary, the basic conditions for productive teams are that the members have defined roles, their work is interdependent, they are skilled and highly motivated, and they work in a trusting environment. To achieve these conditions, there are a number of well-known methods [3, 5, 6, 7, 8, 9, 10]. The team-building principles used by TSP are:

- The team members establish common goals and defined roles.
- The team develops an agreed strategy.
- The team members define a common process for their work.
- All team members participate in producing the plan and they each know their personal roles in that plan.
- The team negotiates this plan with management.
- Management reviews and accepts this plan.
- The team members do the job the way they have planned to do it.
- The team members communicate freely and often.
- The team forms a cohesive group, members cooperate, and they are all committed to meeting the goal.
- The engineers know their status, get feedback on their work, and have leadership that sustains motivation.

Effective team formation requires that the members truly understand what they are supposed to do, agree on how to do the job, and believe that their plan is achievable. These conditions can all be established by involving the engineers in producing their own plans. Then, assuming that their plans are competently made, teams can almost always sell their plans to management.

While all these conditions are necessary for effective teamwork, the specific ways to establish these conditions are not obvious. The TSP provides the explicit training and guidance organizations need to build productive engineering teams.

Launching TSP Teams

TSP projects start with a four-day launch during which the team members make a detailed plan for their project. A trained coach leads the team through determining goals, assigning member roles, making plans, and assessing project risks. By following the PSP planning process and using any available historical data, the engineers are able to make a realistic plan and schedule for their work. Once the team members have built team and personal plans, the team leader holds a management meeting where the team presents the plan to senior management and negotiates an agreed schedule commitment.

Cooperation, Cohesion, and Commitment

While most of the steps in the TSP process involve producing specific things, one condition does not. This is:

- The team forms a cohesive group, members cooperate, and they are all committed to meeting the goal.

These team-member attitudes cannot be legislated, imposed, or established by fiat; they must be created by the team itself. If the team members do not want to cooperate, or if they do not wish to act like a close-knit group, they will not, and telling them to do so will not fix the problem. Commitment is an attitude. When people are truly committed, they behave differently than people who are merely following orders.

While it is clear that cooperation, cohesion, and commitment are necessary, it is not obvious how to produce them. The TSP approach for doing this involves all the team members in producing their own plans and selling these plans to management. People like to work together, they enjoy close-knit and cohesive groups, and they respond to challenging goals and objectives. Unless the working conditions actually block team formation, and as long as the members do not have serious personal antagonisms or emotional problems, such teams will generally become cooperative, cohesive, and committed units.

Team Member Preparation

The first and most fundamental step in the teambuilding process is to ensure that all the members are capable of doing the job. This requires that they have proper skills and abilities and a common set of processes, methods, and terminology. For example, in forming a ball team, you would not pick players from different sports. While they might all be outstanding athletes, a ball team composed of football, baseball, basketball, and soccer players would not likely win many championships. They would not have the common language, agreed rules, or supportive skills to cooperate effectively. Just as on a ball team, TSP team members need a shared process, supportive skills, and the ability to work in an interdependent team environment.

In forming a TSP team, a key requirement is that all the team members understand the principles behind the TSP methods. They must be able to plan and track their work and measure and manage the quality of their products. Training in the Personal Software Process (PSP) provides team members with such knowledge and skill. The major topics that team members need to understand are the following:

- Project planning.
- Status reporting.
- Time, size, and defect measures.
- Quality planning and management.
- Design and design verification.
- Process definition, use, and improvement.

Managers also need to understand these items so they can lead and guide their teams. If the managers are not PSP trained or if the engineers do not know how and why to plan, there is no point in trying to build the group into a cohesive team. The TSP teambuilding process will not work. When the team members do not know how to make plans, they cannot participate in the planning. The team leaders and project managers must then produce the plans. While these may be very good plans, the engineers will not have been involved and they will not be personally committed to the plans.

Conclusions

While the PSP and TSP are relatively new, the experience to date has been promising [11, 12, 13, 14]. The SEI and a growing number of organizations are now qualified to assist industrial and government groups in introducing these methods. There are also an increasing number of universities that teach PSP and TSP courses [2, 15, 16]. For further material on these methods, see the other articles in this issue.

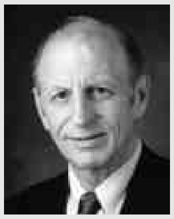
Acknowledgements

The work reported in this paper and several of the other papers in this issue has been developed at the SEI with the support of the Department of Defense. Many people have contributed to this work, but I particularly thank those who reviewed and commented on this paper: Eileen Forrester, Don McAndrews, Jim McHale, Julia Mullaney, Mark Paulk, and Bill Peterson.

References

1. Paulk, Mark C. et al, *The Capability Maturity Model: Guidelines for Improving the Software Process.* Reading, Mass. Addison Wesley, 1995.
2. Humphrey, Watts, *A Discipline for Software Engineering.* Reading, Mass., Addison-Wesley, 1995.
3. Dyer, Jean L, Team Research and Team Training: a state-of-the-art review, *Human Factors Review*, 1984, The Human Factors Society Inc., pp. 286, 309.
4. Shellenbarger, Sue, To Win the Loyalty of Your Employees, Try a Softer Touch, *The Wall Street Journal*, Jan. 26, 2000, page B1.
5. Cummings, Thomas G., Self-Regulating Work Groups: A Socio-Technical Synthesis, *Academy of Management*, vol. 3, no. 3, July 1978, p. 627.
6. DeMarco, Tom and Lister, Tim, *Peopleware, Productive Projects and Teams*. New York: Dorset House Publishing, 2nd. Ed. 1999.
7. Katzenbach, Jon R., and Douglas K. Smith, *The Wisdom of Teams.* Boston, Mass. Harvard Business School Press, 1993, p. 3.
8. Mohrman, Susan Albers, *Designing Team-Based Organizations, New Forms for Knowledge Work*. San Francisco: Jossey-Bass Publishers, 1995, pp. 52, 176, 279.
9. Shaw, Marvin E., Group Dynamics, *The Psychology of Small Group Behavior.* New York: McGraw-Hill, 1981.
10. Stevens, Michael J. and Michael A. Campion, The Knowledge, Skill, and Ability Requirements for Teamwork: Implications for Human Resource Management, *Journal of Management*, Vol. 20, No. 2, 1994.
11. Ferguson, Pat, Watts S. Humphrey, Soheil Khajenoori, Susan Macke, and Annette Matvya, Results of Applying the Personal Software Process, *IEEE Computer*, Vol. 30, No. 5, pp 24-31, May 1997.
12. Hayes, Will, *The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers,* CMU/SEI-97-TR-001.
13. Humphrey, Watts, Using a Defined and Measured Personal Software Process, *IEEE Software*, May 1996, pp. 77-88.
14. Webb, Dave and Humphrey, Watts, Using the TSP on the TaskView Project, CROSSTALK, Vol. 12, No. 2, February 1999.
15. Humphrey, Watts, *Introduction to the Personal Software Process*, Addison-Wesley, Reading, Mass., 1997
16. Humphrey, Watts, *Introduction to the Team Software Process,* Addison-Wesley, Reading, Mass., 2000.

About the Author



Watts S. Humphrey joined the Software Engineering Institute (SEI) of Carnegie Mellon University after his retirement from IBM in 1986. While at the SEI, he established the Process Program, led initial development of the Software Capability Maturity Model®, and introduced the concepts of Software Process Assessment and Software Capability Evaluation. Prior to joining the SEI, he spent 27 years with IBM in various technical executive positions such as management of IBM commercial software development, including the first 19 releases of OS/360. He holds graduate degrees in physics from the Illinois Institute of Technology and business administration from the University of Chicago. He is an SEI Fellow, an ACM member, an IEEE Fellow, and a past member of the Malcolm Baldrige National Quality Award Board of Examiners. He has published several books and articles, holds five patents, and received an award for SPI leadership and innovation from the Boing Corp.

SEI, Carnegie Mellon University
4500 5th Ave.
Pittsburgh, Pa. 15213-2612
E-mail: watts@sei.cmu.edu

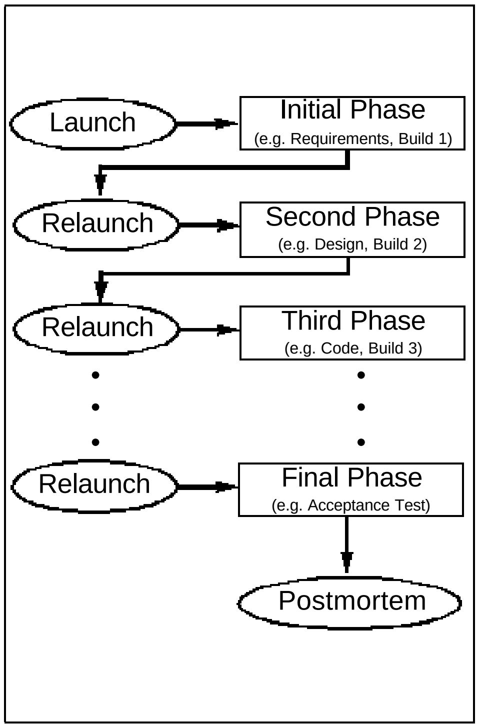
Managing Risk with TSP

by David R. Webb
Hill Air Force Base

One of the most important aspects of applying the Team Software Process (TSPSM) to software projects of any size is the increased success of identifying, tracking, and mitigating risk. The Mission Planning Software Section of the Software Engineering Division of Hill Air Force Base (TISHD), has found the TSP's simple strategy for identifying, tracking, and handling risks to be extremely effective. In fact, many common software project risks are managed purely by adopting the TSP.

Dozens of books address the concept of software risk management and, it seems, there are even more software tools than books on this topic. Risk management tools can be as simple as a list of risks brainstormed during the start of a project and reviewed occasionally. They can also be as complex as a 100-page Risk Management Plan with risks and their associated prioritization, likelihood, impacts and mitigation strategies, along with a Web-based risk browser to track the plan. Still, the basic approach for all of these methods is the same: risks are identified early in the project, planned for, monitored, and handled. TSP takes a middle-of-the-road approach to risk management, doing what makes sense for the project with as little paperwork and tool upkeep as possible. Although the approach was originally designed for teams of fewer than 20 people, the principles can be applied to much larger groups, with equally effective results.

Figure 1. *Risks are identified at each TSP launch.*



Identification

The TSP handles a project the way you eat an elephant—one bite at a time. The TSP team estimates projects in a top-down approach, using overall size and average team productivity to determine overall schedule. This schedule is broken into manageable phases and the phase currently being worked is thoroughly estimated and tracked using a bottom-up approach wherein each engineer estimates his or her own schedule using individual data. Each time a phase begins, whether at the start of the project or at the transition from one phase to the next, there is a project launch (Figure 1). At these launches, the tasks for the current phase are thoroughly defined and each task is estimated using the rigorous methods of the Personal Software Process (PSPSM). These estimates are used to produce a detailed *next phase* earned value plan, against which the project will be tracked and managed. Project goals, quality criteria and risks are also identified during the launches.

A portion of each launch is dedicated to brainstorming risks the project may face. These sessions can last from a dozen minutes to a few hours, depending upon the size of the project and the team's knowledge and maturity. The risks that are identified are serious problems that may occur during the life cycle of the project, not just a list of all maladies that are possible. For example, it makes little sense to manage the risk of your software being destroyed by a bomb or abducted by aliens unless, of course, you work for Special Agents Fox Mulder and Dana Scully. Barring that circumstance, most projects make a list of all the real-life problems that can be foreseen. Some common risks identified during these meetings include a lack of proper documentation, a development environment that may not support the size or type of program being developed, an impossible schedule or inadequate computer, office, or personnel resources. Each risk is assigned a likelihood of occurrence, a severity if it does occur, and a person responsible to monitor the risk. The TSP team assigns each member a role, such as Design Manager, Planning Manager, Implementation Manager, Customer Interface Manager, Quality Manager, Process Manager, Support Manager, Test Manager, or Team Leader. Typically, the team member with the appropriate role is assigned to monitor a risk. For example, a risk involving negotiations with the customer would be assigned to the Customer Interface Manager. This information is documented so that it can be regularly referenced.

Review and Mitigation

The TSP requires a weekly status meeting where team progress is compared to the team plan in terms of earned value and quality. If there are deviations from the plan, the reasons for these deviations can be determined and actions taken to bring the team's performance in line with the plan. It is also during these weekly meetings that the team reviews the risks brainstormed during the launch. The team removes risks from the list that no longer pose a threat, while the assigned engineers report on those that are still potential problems. If the mitigation strategy for a risk has failed and the risk has occurred, or is likely to occur soon, the risk is renamed an "issue" and immediate action is taken to address it. The risk list subsequently becomes a living, breathing document that changes size and shape each week. It also becomes a *used* document that helps the team focus on risks that need to be addressed *when* they need to be addressed.

Risks That Are Not

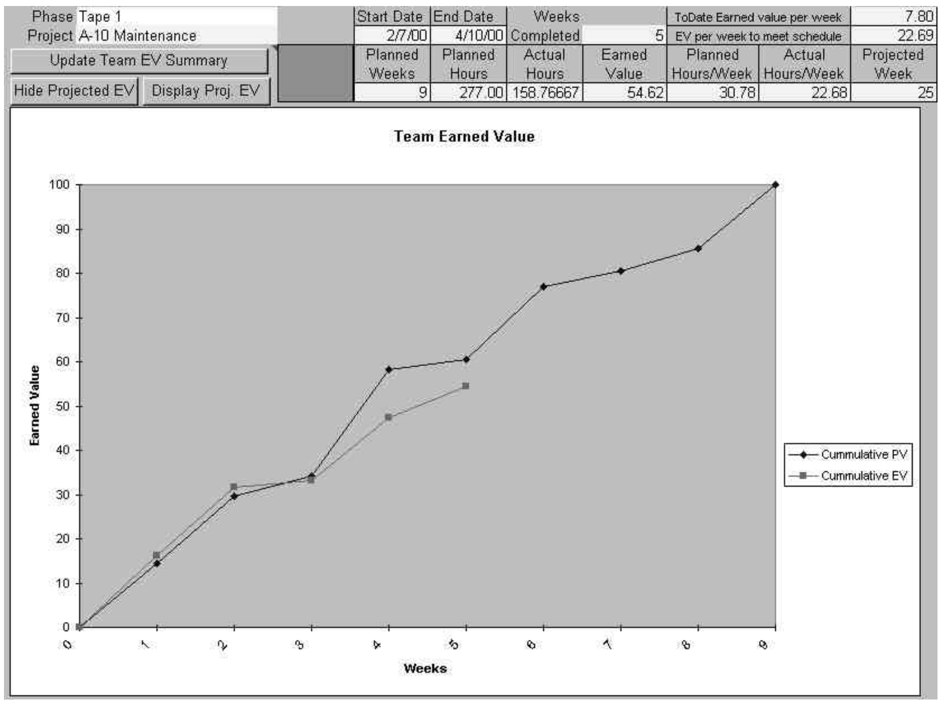
Three of the most common risks to any project are schedule overruns, requirements creep, and quality problems.

A project properly using the TSP already has the tools to handle these risks.

A TSP team determines its own schedule and coordinates it with management, marketing, and the customer, as appropriate. While outside influences may have strong impacts on the delivery date of any piece of software, the TSP team knows its productivity rates, has a rigorous estimating process, and can confidently tell management how much can be accomplished within a given time frame. The TSP launch is not successfully concluded until the team and management agree upon a list of requirements and a schedule that is satisfactory to both parties. Once this realistic schedule has been determined, it is used as the basis for measuring personal and team-earned value and is tracked daily at the personal level and weekly at the team level (Figure 2). Any deviations from the plan are identified early in the project and are dealt with by negotiating with management and the customer. The TSP virtually eliminates schedule risks.

The TSP also requires replanning, or at least updating, a project when the basic assumptions of the plan change. This means that *when* (not if) the requirements change during the course of the project, the team renegotiates schedule, delivered functionality and, if appropriate, cost. This becomes the new plan that the team tracks and the requirements creep risk is effectively dealt with, if not completely eliminated. Another great thing about this technique is that it ensures management and the customer are involved every step of the way so that no one is surprised by the project's performance, least of all those who are anticipating the product.

Finally, problems with quality can, over time, be virtually erased using the TSP. Since the quality methods used by the TSP are based upon the strict quality processes of the Personal Software Process, individual engineers perform their own extensive reviews of both detailed design and code prior to exhaustive team inspections. Defect densities at personal reviews, team inspections, compile and unit test, are used as yardsticks to determine if the finished code is of high enough quality to be passed on to integration and system test, or if the code should be pulled back and reinspected or rewritten. This ensures



Phase	Tape 1
Project	A-10 Maintenance

Start Date	End Date	Weeks Completed	
2/7/00	4/10/00	5	

ToDate Earned value per week	7.80
EV per week to meet schedule	22.69

Planned Weeks	Planned Hours	Actual Hours	Earned Value	Planned Hours/Week	Actual Hours/Week	Projected Week
9	277.00	158.76667	54.62	30.78	22.68	25

Figure 2. *TSP Team Earned Value for TISHD A-10 AWE*

the quality of the code, but not always the quality of the requirements upon which the code is based. Often, requirement problems are uncovered during acceptance testing. When such defects are discovered, the TSP team adds them to team and personal review checklists to ensure such problems are never allowed to pass through the process again. An experienced TSP team can, therefore, eliminate virtually all quality risks, particularly expensive defects found during qualification and acceptance testing.

Some Examples in TISHD

TISHD has trained nearly 20 engineers in the PSP and has launched three separate mission planning projects using TSP version 0.3. These projects are an Air Tasking Order parser named TaskView [1], an A-10 Aircraft/Weapons/Electronics (AWE) software program, and an F-16 Block 30 AWE program. Of these three projects, TaskView and A-10 AWE have been using the TSP long enough for us to draw some conclusions about the usefulness of the TSP and the success rate of using the TSP risk management strategy.

TaskView

The TaskView project was the first TISHD group to pilot test the TSP. Just prior to the initial launch, the TaskView customer decided to participate in an Air Force Expeditionary Force Experiment (EFX). This new goal required the TaskView 3.0 product to be delivered one month earlier than originally planned. The team added this risk to its risk list and assigned it to the Planning Manager. With this in mind, the team adjusted the plan to meet the new schedule.

As work progressed, the team-earned value projected that TaskView 3.0 would be delivered more than a month earlier than anticipated, even with the new schedule. At this point, the first risk was closed out and another risk—that of being too early and losing revenue—was added to the list and assigned to the Customer Interface Manger. The customer was approached with the option of receiving the product early and getting a refund, or adding new capability to TaskView 3.0. The customer was delighted with this information and chose to keep the current level of funding and add in new capabilities to the software. Even with the new functionality, TaskView 3.0 was delivered well within time to participate in the EFX experiment.

TaskView has also experienced a significant reduction in the risks associated with defects, as a result of adopting the TSP. TaskView has had three major releases since TISHD started working on the project in 1997. TISHD has added new capability and robustness to each release, at times rewriting major portions of the code to do so. Compared to data from similar projects completed in the past (using the TISHD CMM Level 5 organizational process), the TaskView projects have seen a substantial decrease in defects and test time (see Figure 3 and Figure 4).

One interesting outcome of the defect data analysis was the increase in defect density experienced by the TaskView 3.1 project. Although the defect density found during Customer Acceptance Testing was steadily decreasing, defects found in earlier test phases increased. This was of some concern to the team, until it began to filter the list by defect priority (Figure 4). Once that was done, it was obvious that the TaskView team, as it had grown more confident in the use of the TSP, had begun to record more development defects than ever before; remember, TSP teams count every defect found in every development and test phase, including compile. However, despite this increase in defect recording, high-priority defects became nonexistent using the TSP. This does not mean that no issues were discovered during customer acceptance testing, but the issues dealt almost exclusively with the addition of new requirements and limitations of the operational environment, and were not defects in the delivered code. Note also that TaskView 3.1S (a special project developed in support of another mission planning tool) had zero high-priority defects at *every* test phase.

As most software project managers are well aware, the greatest risk to any project's schedule is the risk of finding defects during test, especially final or customer acceptance testing. The causes of these defects are often difficult to trace and fix and can cause significant slips in schedule. In order to eliminate this risk, test time needs to be reduced and become more consistent. Although TISHD was already seeing very low test days/thousand lines of code rates using its Level 5 process, the adaptation of the TSP reduced the test time further and made the variation much smaller (see Figure 5).

Figure 3. *TISHD Total Defect / Non-TSP Projects vs TaskView*

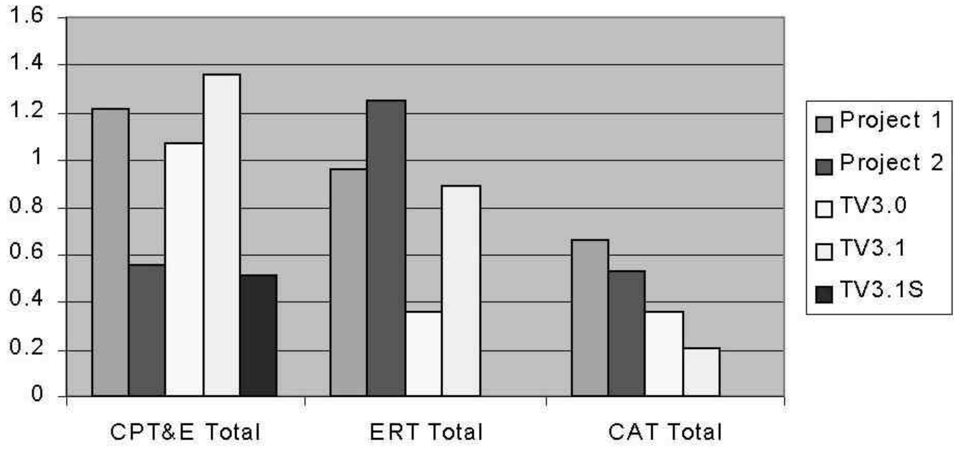
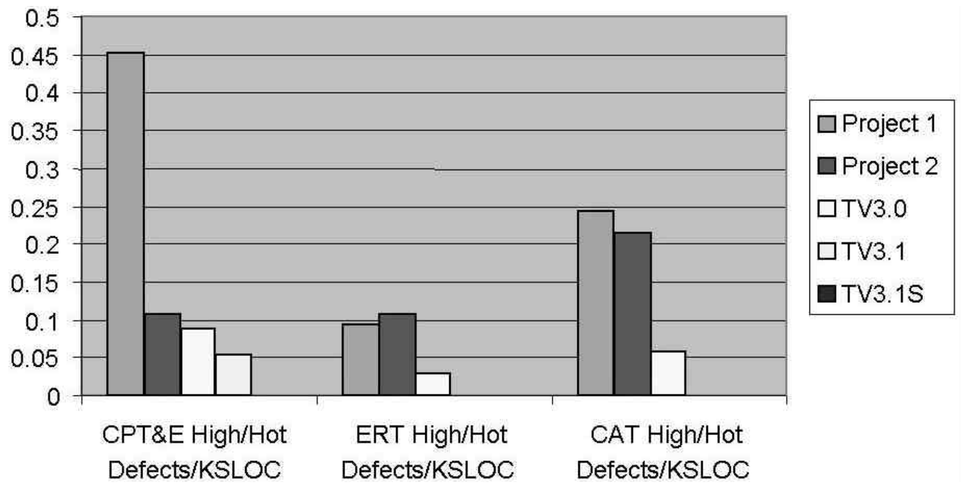


Figure 4. *TISHD High Priority Defects / Non-TSP Projects vs TaskView*



A-10 AWE

While the TaskView project was still evaluating the effectiveness of the TSP, the A-10 AWE team decided to use some of the concepts (planning, tracking, weekly updates) *without* employing the rigorous techniques of the Personal Software Process. Each A-10 AWE engineer was provided a spreadsheet for each code change he or she was working. These spreadsheets covered the estimate of size (lines of code or LOC) and time (days) as well as the actuals for LOC and time. Time was measured at distinct milestones, such as inspections, unit test, and code check-in. An earned value plan was created from the estimates provided by the engineers and used to refine the schedule. Although the engineers were not required to be PSP trained, all any engineer had to do, after estimating, was to check a box on the tracking spreadsheet once a milestone was reached. The spreadsheet would calculate how long the tasks took and export that data to the earned value tracking tool.

Sounds like a good plan, right? It did not work very well.

Estimates were often wildly inaccurate. Tracking was not consistent. Entire new capabilities would move from 0 percent complete to 100 percent complete overnight. All of these problems gave the team a false impression of the team-earned value. The earned value was, therefore, not trusted and soon ignored by most of the team members. The team reverted to the higher-level tracking process used by non-TSP projects in TISHD, which were sufficient to prevent the team from missing schedule. (Note that TISHD is part of a SW-CMM® Level 5 organization, and typically meets cost and schedule estimates anyway.) However, any advantage of using the TSP-like process disappeared.

The one thing that *did* work, was risk identification and tracking, the process that we copied directly from the Team Software Process.

For example, the A-10 AWE team determined that a required piece of core software, developed by a third-party vendor, might not be released in time to meet schedule. This risk was assigned a high likelihood and a high impact. A mitigation plan of reverting to an earlier release of the core was determined and an engineer was assigned to track the status

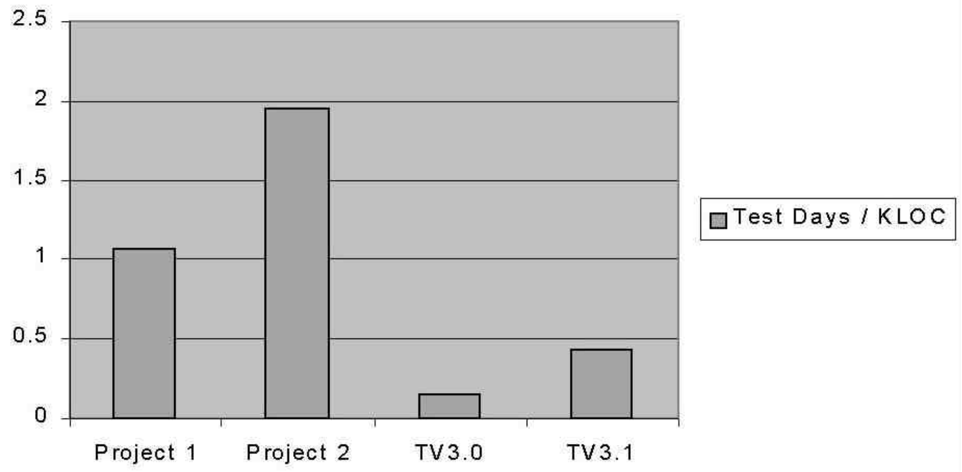


Figure 5. *TISHD Test Duration / Non-TPS vs TaskView*

of the core software. As it turned out, the third-party software did slip its schedule by several months, which would have, in turn, caused our software to slip its release date had we not planned for this risk early in the program. Due to early risk identification, planning and tracking, the A-10 AWE was able to mitigate this risk and revert to the earlier version of the core software.

One risk that was *not* identified was the hazard of using the TSP-like process, instead of the TSP. During project post-mortem, it was determined that the reason the modified process did not work as well as a traditional TSP team was that the engineers were not PSP trained and did not understand how the data they were collecting was being used. At that point, we determined to use TSP on the next A-10 AWE project and immediately scheduled a PSP course for those engineers.

The results in earned value tracking alone were astounding (see Figure 3). Code was accurately estimated and tracked; it was *very* easy to see how close to our schedule we were running. TISHD learned an important lesson: TSP does not work well without the proper data, and that data is almost impossible to gather without the rigors of the PSP. That is one risk TISHD has completely eliminated.

Conclusion

While there are many tools for software risk management, TISHD has found that utilizing the planning, tracking, and defect prevention techniques of the Team Software Process is a simple and effective way to identify, track, and mitigate most software project risks. In TISHD we have learned that, over time, TSP teams become experts at risk mitigation and management; they also become very good at writing code that is nearly free of defects, and that TSP is a risk mitigation strategy *any* software project should strive to adopt.

Reference

1. Webb, David and Humphrey, Watts S. Using the TSP on the TaskView Project, CROSSTALK, February 1999, pp. 3-10.

About the Author

David R. Webb is a Technical Program Manager for the Mission Planning Software section at Hill Air Force Base, Utah, and a part-time visiting scientist for the Software Engineering Institute (SEI). He is a member of the Software Division of the Technology and Industrial Support Directorate (TIS), which was assessed a CMM® Level 5 organization in July 1998. He has 12 years of technical and program management experience with software in the Air Force. Webb also has spent five years as a software test engineer, two years as a software system design engineer, and three years as a member of TIS's full-time Software Engineering Process Group (SEPG). He is a SEI-certified PSP instructor. He received a bachelor's degree in electrical and computer engineering at Brigham Young University.

OO-ALC/TISHD
6137 Wardleigh Road
Hill Air Force Base, Utah 84056
Voice: 801-775-2916 DSN 775-2916
E-mail: david.webb@hill.af.mil

Coming Events

June 4-7
9th Biennial IEEE
http://cefc2k.aln.fiu.edu

June 4-11
22nd International Conference on Software Engineering
www.ul.ie/~icse2000

June 5-7
2000 IEEE International Interconnect Technology Conference
www.his.com/~iitc

June 10-14
ISCA2000: 27th International Symposium on Computer Architecture
www.cs.rochester.edu/meetings/ICSA2K

June 18-22
ICC 2000—IEEE International Conference on Communications
www.icc00.org/

July 11-13
5th Annual Conference on Innovations and Technology in computer Science Education
www.cs.helsinki.fi/events/iticse

July 16-18
7th IEEE Workshop on Computers in Power Electronics
www.conted.vt.edu/compel.htm

July 16-19
Congress on Evolutionary Computation
http://pcgipseca.cee.hw.ac.uk/cec2000

August 6-11
6th Annual International Conference on Mobile Computing and Networking
www.research.telcordia.com/mobi-com2000

August 7-8
IEEE Workshop on Memory Technology Design and Testing
http://pcgipseca.cee.hw.ac.uk/cec2000

August 17-19
Designing Interactive Systems (DIS)
September 10-12
Collaborative Virtual Environments (CVE)
September 10-14
Very Large Databases (VLD)
Visit www.acm.org/events for information on VLD, DIS, & CVE 2000.

April 29-May 3, 2001
STC 2001: The Premiere Department of Defense Software Technology Conference
www.stc-online.org

Making Quality Happen: The Managers' Role

Girish Seshagiri
Advanced Information Services (AIS)

When managers insist that their professional employees rigorously apply the recognized disciplines of their fields, they will do better work [1]. In this paper, we provide data from two AIS projects, Project A and Project B, that demonstrate how managers profoundly affect the way their engineers behave and how managers can motivate the engineers to apply the disciplined methods they have learned. Project B data is used here as control data. We conclude with lessons learned.

Advanced Information Services Private Ltd. is the Indian subsidiary of Advanced Information Services Inc. of Peoria, Ill. This case study is based on data from two projects executed by the subsidiary company in Chennai, India. Engineers in both projects have been trained in the PSP^{SM}. They used PSP methods to plan the critical design/implementation/test phases of the projects.

Project Domain, Technical Environment, and Engineers' Qualifications

The projects' mission is to build Personal Productivity System (PPS), a commercial tool to automatically log time, track defects, maintain data, do calculations, and simplify routine tasks. Personal Planning Assistant (PPA 1.0), and Personal Quality Assistant (PQA 3.0) are two subsystems described in this article.

The target environment is a two-tier client server architecture with a Visual Basic client application for Windows 95, 98, NT, and SQL Server (under Windows NT) for the server.

The engineers' experience level ranged from one to two years. Project managers had three to five years' experience. All have a master's degree in computer science or computer applications. All are trained in PSP, software inspections, managing the software process, and requirements engineering—the required software engineering training for AIS engineers.

Development Strategy

Following the requirements and high-level design phases, PPA 1.0 had 13 components to be developed and PQA 3.0 had four components.

PPA 1.0 was divided into three incremental development phases. A project manager and three engineers participated in Increment 1 development. Two more engineers were added for Increment 2. A project manager and three engineers participated again in Increment 3 development. PQA 3.0 team consisted of a project manager and three engineers.

All engineers were responsible for creating PSP plans for their components. Project managers also participated in development.

The sizes of all 13 modules of PPA 1.0 are given in Table 1.

PPA1.0 Increments 1 and 2 – Delivery Commitment *What must happen* and *What will happen*

The Chennai team had committed to deliver PPA 1.0 (Increments 1 and 2) to the AIS Development Group in the U.S. by Aug. 1, 1999—four weeks prior to the SEI Symposium. The AIS Development Group had planned to demonstrate the PPS product (PPA1.0 Increments 1 and 2 and PQA3.0) at the Symposium.

PPA 1.0 Increment 1 was completed four weeks behind schedule against a planned schedule of 16 weeks. Analysis of the Increment 1 Defects by Process chart (See Figure 1.) showed that engineers found and fixed more defects through team inspections and test than through personal reviews.

Planned test defects/KLOC were high for PSP trained engineers; actual test defects/KLOC were even higher (see Table 2).

About half of the slippage in schedule was directly the result of engineers spending more time than planned in test and rework. Their plans simply did not include enough time for early defect removal through personal design reviews, code reviews, and team inspections.

The engineers' PSP Plan Summary indicated that they had spent time on post-mortem when their modules were completed. Asked what they did in the post-mortem phase, the engineers said that they only had time to gather the PSP data for inclusion in the organization database, and that they did not have time to analyze the data and adjust the plans for Increment 2.

As the acting Development Manager, I reviewed the plans for Increment 2. I realized that the plans were based on the team's perception of what must happen (i.e. we *must* ship by August 1). It was obvious that if the team planned Increment 2 similarly as in Increment 1, what would happen is that the schedule would slip again and we would likely miss our dates.

Engineers Not Using Known Effective Methods

Clearly, the engineers were not using the disciplined methods

Table 1. *Module Size*

Module Name	Module Size (Lines of Code)
Estimate Size	2107
PROBE	2043
Plan Summary 1	1738
Plan Summary 2	1154
Track Time	1914
Track Size	4250
Size Range	965
Object Data	1123
Standards & PQA Update	1286
Interruptions & Tool Bar	403
Defect Analysis	850
Plan Analysis	532
Quality Analysis	714

Table 2. *Defects/KLOC*

Test Phase	Plan	Actual
Unit/Integration/System	4.8	6.2
Acceptance	0.7	1.4

they learned in the PSP training. They knew that the PSP Plan Summary gave them useful data on their performance and they should spend time in post-mortem to improve the process and set personal goals.

They knew that they should strive for at least 80 percent yield, and Appraisal/Failure Ratio (A/FR) should be greater than 1.5 and close to 2.0. [2]. Yet, their planned A/FRs were less than 1.0. (See Table 3 for definition of PSP quality measures.)

The engineers were also aware of AIS Chennai's business goals of defect-free delivery of the PPS product on time.

Changing the Engineers' Behavior

I realized that I must direct the change in the engineers' behavior. I must insist that they rigorously apply known PSP principles in their work. If I did not, nobody else would. In a team meeting I reviewed the PSP data for size, appraisal hours, failure hours, and test defects for each of the four modules in Increment 1. I used charts showing relationship of yield and Lines of Code (LOC)/hour (Figure 2) and A/FR and test defects. (Figure 3). It was not possible to draw statistically valid conclusions with only four data points. It was not necessary. The engineers got my message. Schedules were important. But a quality process is how we measure our success.

Goal Setting

I involved the project manager and engineers to set process quality goals for Increment 2:

Goal 1: Reduce test defects/KLOC
Plan for A/FR between 1.5 and 2.0

Goal 2: Increase effectiveness of design and code reviews
Plan for reviews of 100–150 LOC/Hour

Goal 3: Increase process yield
Plan for yields greater than 80 percent

PPA1.0 Increments 2 and 3 – *Quality is free*

The engineers revised their plans. They planned for more appraisal time and less failure time. They increased their post-mortem time for analysis and process improvement. They increased total development staff hours by less than 10 percent.

Table 3. *Definition of PSP Quality Measures*

Measure	Formula
Failure Hours	Total time spent in test (compile time not applicable in Visual Basic)
Appraisal Hours	Total time spent in design and code, personal reviews and design, and code team inspections
Failure Cost of Quality	100 × (Failure hours)/Total development hours
Appraisal Cost of Quality	100 × (Appraisal hours)/Total development hours
Total Cost of Quality	Appraisal COQ + Failure COQ
A/FR Ratio	Appraisal COQ/Failure COQ
Overall Process Yield	100 × (defects removed before test)/(defects injected before test)
Personal Review Yield	100 × (defects removed in personal reviews)/(defects removed in personal reviews + escaping from personal reviews)

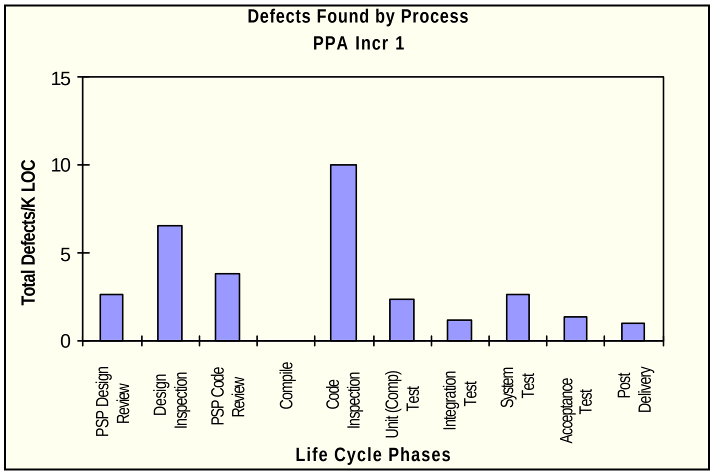


Figure 1

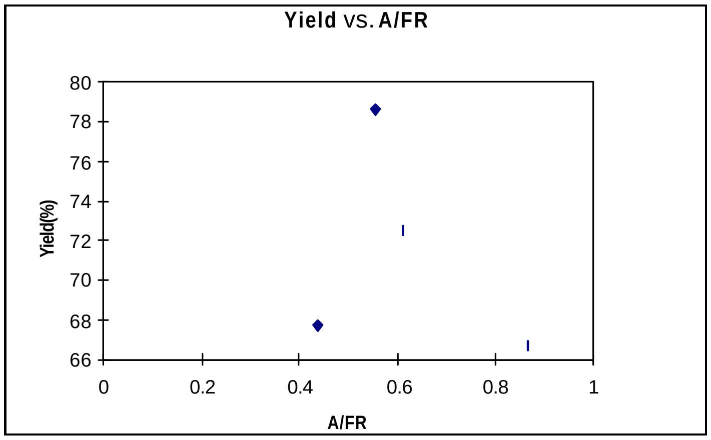


Figure 2

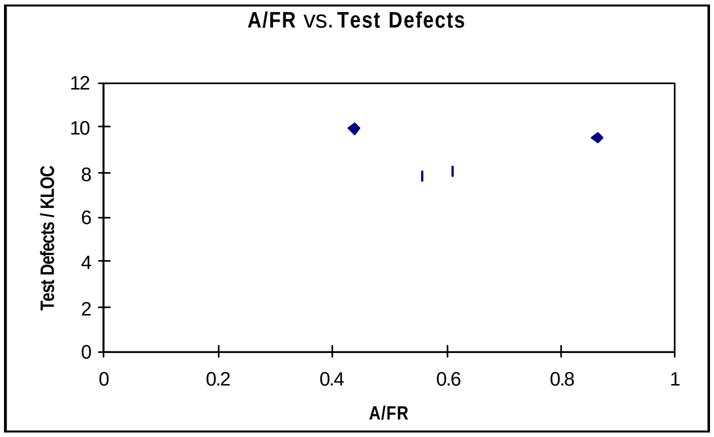


Figure 3

Increment 2 was completed on schedule. The team analyzed the data at the conclusion of Increment 2. Process data showed improvement in measures such as yield, A/FR, and test defects/KLOC. Cost of quality remained nearly the same.

The team set more aggressive goals for Increment 3. They planned for A/FRs greater than 2.0, process yield greater than 90 %, and test defects/KLOC less than 1.0. Process data showed further improvements in quality measures. Cost of quality declined.

We present the results in charts that accompany the Web version of this paper. The data in the charts are averages of four modules in Increment 1, six in Increment 2, and three in Increment 3 of PPA 1.0 and four modules in PQA 3.0.

PQA 3.0 is used as control group since it did not have the same executive leadership to direct and change engineers' behavior by defining criteria for success and setting aggressive individual and team goals.

Lessons Learned

1. When engineers use PSP on a real project following classroom training, their plans continue to rely on what they are most comfortable with. More time for test and rework and less time for reviews and inspections for early defect removal.
2. Management support is most critical during the transition from classroom to actual industrial use of the PSP.
3. Management support should include active participation in data analysis, goal setting, and process improvement.
4. Schedules dictate what must happen. Engineers' personal data show what will happen.
5. Managers and engineers should jointly make commitments based on what will happen and learn to manage by data.
6. For incremental development to be effective, managers and engineers should spend post-mortem time to adjust and improve the process.
7. PSP Cost of Quality measure provides compelling evidence that quality is free.
8. PSP enables alignment of engineers' personal goals with business goals for defect-free software delivery on time.
9. For quality to happen, managers and engineers must have mutual trust.
10. Engineers tend to improve their performance over time when they use a disciplined process and management is supportive.
11. We still have a long way to go to realize the full human potential in software development.

Acknowledgments

Special thanks to the AIS Chennai software engineers Antony Sudhakar, D. Giridharan, R. Kailasam, K. Manicavel, Paul Jaison, R. Suresh B. Sivapriya and AIS Chennai project managers M. Jeyalakshmi, and S. Srinivas. Thanks also to R. Soudarsanan in Chennai for document preparation and Rafiuddin Syed in Peoria for review of the draft.

References

1. Humphrey, Watts S. *Managing Technical People,* Addison-Wesley, Reading, MA, 1997.
2. Humphrey, Watts S. A *Discipline for Software Engineering,* Addison-Wesley, Reading, MA, 1995.

Please refer to the Web version of this paper [available at www.stsc.hill.af.mil] to see the complete set of module charts.

☞The Software Quality Institute in Chennai, India was recently named after Watts S. Humphrey.

About the Author

Girish Seshagiri is the CEO of Advanced Information Systems Inc., a winner of the 1999 SEI/IEEE Computer Society Software Process Achievement Award. He is also the acting Executive Director and co-founder of The Watts Humphrey Software Quality Institute (SQI) located in Chennai, India. He received his master's of science degree in physics from the University of Madras and his master's degree in business administration in marketing from Michigan State University.

Advanced Information Services Inc.
1605 W. Candletree Drive, Suite 114
Peoria, Ill. 61614
Voice: 309-691-5175, ext. 217
Fax: 309-691-5440
E-mail: girish@advinfo.net

PSP/TSP Web Sites

http://stsc.hill.af.mil/ptech/pt_art.asp
This site by the Software Technology Support Group features its Process Team's favorite CrossTalk articles. It links readers to such articles as *Process Assistance Visit: A Tool for Process Improvement; Software Process Automation: A Technology Whose Time has Come; Air Force Policies on Attaining SEI CMM Levels;* and *Continuous Process Improvement for Software: Data Definition.*

www.sei.cmu.edu/publicaitons/documents/97.reports/97tr001/97tr001abstract.html
Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers. This report documents the results of a study that examined the impact of the PSP on the performance of 298 software engineers. The report describes the effect of PSP on key performance dimensions of these engineers, and discusses how improvements in personal capability also improve organizational performance in several areas.

www.cs.usak.ca/grads/vsk719/academic/856/project/project.html
T*he Personal Software Process in Meta-CASE CMPT 856 Project* by Vive S. Kumar has links to PSP-related topics such as an Overview of PSP Principles of Meta-CASE Systems, How to Incorporate PSP in Meta-CASE, and Metrics for PSP.

http://archives.distance.cmu.edu/psp/pre_May
The Personal Software Process^{SM}: A Practitioner's Starter Kit is a course intended for practicing software engineers and their managers. It introduces the highest-leverage metrics of PSP. Students watch nine lectures, do seven programming problems, four reports, and read selected chapters from Watts Humphrey's book, *A Discipline for Software Engineering.*

http://psp.distance.cum.edu/oct/resource/online.html
This site offers a list of online readings related to PSP.

www.sei.cmu.edu/psp/Results.htm
This is a Software Engineering Institute site with links to sites such as Defects vs. Experience (programs 1 and 10), Yield vs. Program Number, and Lines of Code per Hour Improvement.

http://emhain.wit.ie/!doconnor/lectures/se3/project/team14/slides/psp1.htm
This gives an introduction and the seven progressive steps of PSP, such as Baseline Personal Process, and Personal Quality Management.

www.computer.org/computer/co1997/r5924abs.htm
This is an article by Pat Ferguson, Watts Humphrey, Soheil Khajenoori, Susan Macke and Annette Matvya that appeared in the May 1997 issue of *Computer* magazine. The article is "Results of Applying the Personal Software Process."

www.acm.org/pubs/citations/proceedings/cse/273133/p322-hou/
This takes visitors to the proceedings of the 1998 SIGCSE technical symposium on computer science education, and to the paper, "Applying the PSP in CSI: An Experiment."

PSP: Fair Warning

by Elizabeth Starrett
Air Force Mission Planning System

The Personal Software Process (PSP)SM Course is difficult and time consuming. I started working in a group that had already been trained on PSP and was using the Team Software Process (TSP)SM. I attended the PSP training with a second group from my section that had not had the training and was supposed to start implementing PSP and TSP on its upcoming project. This article discusses some of the difficulties the students experienced during and after taking the course. It also discusses the results experienced by my group already using TSP and the hope that life can get better.

So, you want to take a course on the Personal Software Process (PSP). Or, you do not want to take a course on PSP, but your management is forcing you. Either way, there are some things you should know before taking the course.

Definitions

PSP is a combination of personal software development processes as well as data collection and use suggestions intended to help the programmer develop software with better quality and predictability. TSP is a series of processes that incorporate the ideas of PSP into a team environment.

Prerequisites

First, make sure you are familiar with several basic functions that your development system provides. You will need to know how to code or call typical math functions such as square roots and natural logarithms. You will also need to be able to create, open, close, and access files; write to the computer monitor, and read from the keyboard. Make sure you know how to create and use linked lists. Using linked lists is a requirement dropped by some instructors who allow students to use arrays instead. Not all teachers are so lenient; mine was not. There is also a small group of people who prefer self-torture and will not use the equations provided by the teachers on blind faith. If you fall into this group and need to understand and believe the equations before using them, your preliminary knowledge will also require a basic understanding of statistics and its notation and calculus' concept of numerical approximation to integration.

Lots of Time

The next point—and probably the biggest—is the amount of time the course will take. There are different ways of formatting the course. The course is usually taught in two, one-week sessions. The first week of this format requires about 4.5 hours of lecture each day and the development of six programs, a coding standard, a Line of Code (LOC) counting standard, and a mid-term report. The second week requires about 4.5 hours of lecture each day as well as the development of four programs and a final report for the course.

If you are lucky enough to arrange a more lenient format, the homework will be the same, but the lecture time might be reduced to about three hours per day and there will be more days available to do the homework. My class was broken into four sessions. The first session was three days, the second and third sessions were four days, and the fourth session was three days (14 days vs. 10 days).

Maybe you are the brightest person and best programmer in your group. If you are, you can probably plan on attending the lectures and completing most of your programming homework in close to the same amount of time you would typically spend at work. You will probably need extra time for developing the coding standards, writing the sixth program, and writing the reports. One of my colleagues, whom I would put in this category, estimates he spent about eight total extra hours.

If you are one of the average programmers in your group, the amount of extra time you spend at work will depend on the amount of time your instructor spends on the lectures. Our lectures typically took about three hours each day. The time spent on just the 10 programs was an average of 62.6 hours per student. Add an average of 6.26 hours for each program to the lecture and you are putting in a lot of extra hours. Bear in mind that this documented time is only the task time and does not include times for interruptions such as taking phone calls, using the rest rooms, getting a drink, etc. You can typically plan on doubling the task time to get the overall time spent. Some programs are harder and some easier than others are, so the time you spend each day will vary. I was an average student and I spent about 30 extra hours working on the programs. However, I was not average on my time spent for the reports. My background is in the metrics arena, so I spent more than average time looking at all my data and trying to draw different conclusions for my reports. My mid-term report took about six total hours and the final report took about 20 total hours. (I also had 14 days to do this. If your course is taught in 10 days, the amount of extra time required will probably increase by about 32 hours.)

The programmer who has not been doing much programming lately or just is not that comfortable with it might want to avoid this class. Instructors have told me that they have had classes where at least one student spent months trying to develop the programs. The homework is not easy.

Implementation

After the course is over and the class has graduated, the process does not roll simply into the work environment. My class finished in November 1999, and four months later most of the students were still struggling to implement PSP and TSP into their newest project. My organization even has an internal tool that we developed to help automate the process, but the tool often creates as many headaches as the manual process. There have been numerous discussions on how work breakdown structures should be organized and how they should be used with the tool. Also, the tool merges with another SEI PSP tool and between the two of them, data is often lost and/or misrepresented.

Without TSP, PSP is not likely to be institutionalized. Six months after PSP was first taught in my organization, the instructor called all the students to see who was using it. Only one student was still using it. When the instructor called SEI asking for suggestions on how to institutionalize PSP, my current work group became a beta test site for TSP. The test results were impressive; both PSP and TSP are institutionalized in my group.

Results

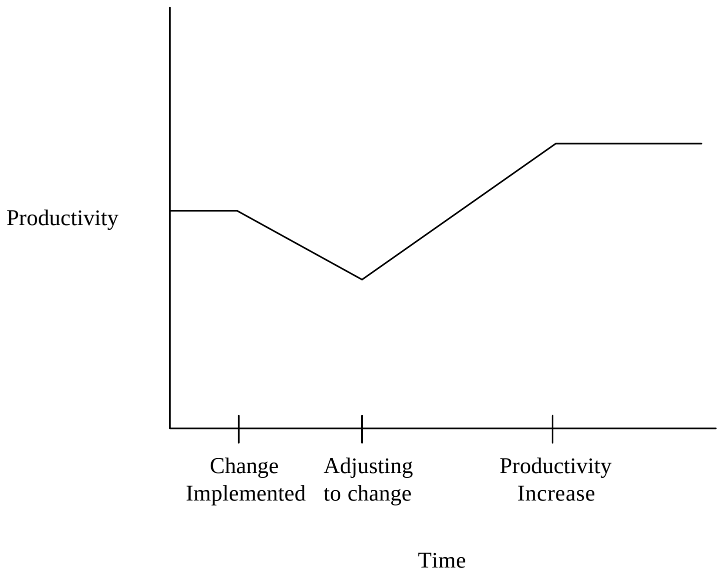
After all this struggle and all this work, do the PSP and TSP processes really help? Opinions vary. Several in the group that took the class with me agree the process portion of PSP is useful, but they do not agree all the data are useful and certainly not worth the effort to collect and try to use. This might be because they are not used to using the tools and will become accustomed to them with more time. Then again, they might not.

Figure 1 is a technology adoption curve that shows productivity decreases as any change is implemented. Learning a new way of doing things takes time. However, the theory is that a worthwhile change will cause productivity to increase to a point that overcomes the initial loss in productivity. Will the group now starting PSP make great and glorious gains? I do not know.

My work group is one of the first to implement the TSP and it received the division's quality award for the excellent results of the software released on budget, on schedule, and with minimal defects found in test. My group believes in using PSP, but opinions differ on the usefulness of some of the data. As time goes on, we may stop collecting some of the PSP measurements that we are not finding useful. My group agrees that the data collection process needs some sort of automation because a manual process is too cumbersome.

While the initial implementation of PSP is expensive, there is the hope that maintaining the PSP process will cost much less. Before coming to work in my group, I had already taken the PSP Executive Course (about eight hours) and understood the PSP concepts. I was able to walk into my current group and start using its process, with a few minor errors along the way. As I took the full PSP course, it became obvious that since I had already had the Executive Overview (so I understood why I did what I did) and I was already living the PSP process, I really did not need this course. I could have acquired the same knowledge with a brief course on the size- and time-estimating method and tool used by PSP (PROxy-Based Estimating [PROBE] method). After an organization's initial training and adoption into daily business, new people can be given the PSP executive overview, about two hours PROBE training, and a written process to follow to have what they need to use PSP with the rest of the group. This is assuming the rest of the group outnumbers the new people, so the experienced people can reasonably provide assistance. Maintenance has a much lower cost than the initial implementation.

Figure 1



Conclusion

I came to work in my current group after not developing software for 6.5 years. I came with an open mind and a desire to learn how to develop software using Level 5 processes. I was provided a written script of my group's software development process and support from my team. As a result, I have enjoyed developing code here much more than with my previous organizations and I have much more confidence in the code I am releasing to our customers. My opinion is that we are doing wonderful work, our customers are happy, and PSP is worthwhile; but I did not pay for it either.

Notes

1. The automated PSP tool developed and used by my organization is open source and freely available to anyone interested. The only payments requested are feedback from the users and the sharing of any improvements made to the software. The point of contact for further information on this tool is Ken Raisor (E-mail: ken.raisor@hill.af.mil).
2. There is a third group in my section that was already operating at a Level 5 before taking the PSP course. After taking the course, the group elected to continue using its old process instead of adopting PSP. However, the group's development processes are very similar to PSP.

About the Author

Elizabeth C. L. Starrett has been a member of the TaskView development team for the past year, supporting the Air Force's Mission Planning System. Prior to this, she was a software process improvement consultant for the Software Technology Support Center (STSC), where her duties included leading the STSC Measurement Team. Starrett has spoken at the Software Technology Conference, and at the Data Reduction and Computer Group Conference, and been previously published in CrossTalk. Prior to joining the STSC, she developed, documented, and tested data analysis and test support software for radar and the Peacekeeper missile, working for the Air Force and its supporting contractors. Starrett has a bachelor's degree in electrical engineering from Utah State University.

OO-ALC/TISHD
6137 Wardleigh Road
Hill AFB, Utah 84056
Voice: 801-775-2838 DSN 775-2838
Fax: 801-775-2541 DSN 775-2541
Internet: beth.starrett@hill.af.mil

Statistical Process Control Meets Earned Value

by Walt Lipke and Jeff Vaughn
Oklahoma City Air Logistics Center

Levels 4 and 5 of the Software Engineering Institute Software Capability Maturity Model (SEI CMM®) imply the application of Statistical Process Control (SPC) to software management. SEI staff members have published a book [1], and are teaching a course [2] on the subject. Several software organizations are trying to apply SPC to quality control. This article expands the area of application. It presents an approach for software production management (i.e., cost and schedule control.)

The Test Program Set and Industrial Automation Branches of the Oklahoma City Air Logistics Center, Directorate of Aircraft Management, Software Division achieved Level 4 of the SEI CMM®)on November 7, 1996. At that time, and continuing today, this software group applies several measures in the control of its process and product output. The measures relate to financial health, project management, workload and labor, and process improvement (rework and productivity). Rudimentary SPC has been applied to the measures for some time in the form of run charts [1 and 3] (i.e., charts that graphically portray measured results in chronological sequence).

Run charts are fine, they provide trend information. But in the traditional quality application, what is expected are control charts [1 and 3]. From classes [2 and 4], conference presentations [5], and books [1 and 3], it is implied that *you are not really applying SPC unless you are using control charts.* Here is where *six sigma* originates. The predominant thought is you cannot have *six sigma* (translation: *really good*) quality unless you know the process is in control. Understanding whether or not the process is in control comes from the use of control charts, thus, the impetus to apply this SPC technique to software quality control. It is our understanding that the software organizations attempting to apply SPC are using data taken from product reviews, predominantly directed to the coding portion of the process. They are using defects identified in relation to effort expended, or lines of code or function points as data for the control charts.

The application of SPC Control Charts in this article does not focus on analysis of software defects. The following discussion will illustrate how SPC can be coupled to Earned Value indicators to provide information about the quality of project performance. The use of SPC Control Charts then becomes a tool in the control of software project cost and schedule.

Earned Value

The indicators from Earned Value (EV) Management, which directly relate to efficiency of project execution, are the Cost Performance Index (CPI) and the Schedule Performance Index (SPI). Their definitions are:

CPI = BCWP/ACWP (where BCWP is the budgeted cost of work performed, and ACWP is the actual cost of work performed).
SPI = BCWP / BCWS
(where BCWS is the budgeted cost of work scheduled).

For additional information and explanation concerning these formulas and terms, please refer to [6].

These two indicators, taken together, can be used to manage project performance [7]. They can provide very insightful information for managers regarding the status of their project. As described in the referenced article [7], when the inverse values of CPI and SPI (CPI^{-1} and SPI^{-1}) are compared to their respective cost and schedule ratios and the results are paired, one of nine recommended management actions is determined (see Table 1). As discussed in the article [7], the management actions are related to four possible strategies:

- Adjusting overtime or number of employees.
- Realigning employees to increase efficiency.
- Reducing performance requirements.
- Negotiating additional funding or schedule.

With this background, the Earned Value indicators, CPI^{-1} and SPI^{-1}, were chosen for SPC application. One very good feature of these EV indicators is they are *normalized.* Regardless of software project conditions (e.g., size of project, experience of staff, software engineering environment, programming language, etc), their ideal value is 1.0. Because they are *normalized*, many of the issues with applying SPC to software, such as variability and homogeneity of the data, are avoided.

Statistical Process Control

Software project managers normally assess their project status on some periodic basis. In our organization, we perform project reviews monthly. The project data for CPI and SPI is aggregated from the individual developers, then computed, charted and analyzed monthly along with several other indicators. Because the set of project data for analysis of each indicator (CPI^{-1} and SPI^{-1}) has only one data point per month, the type of SPC Control Chart selected is XmR, or individuals and moving range [1 and 3].

Table 1. *Management Actions*

CR vs. CPI^{-1}	SR vs. SPI^{-1}	Management Actions
Green	Green	Reward Employees
Green	Yellow	Increase OT
Green	Red	Increase OT or People
Yellow	Green	Decrease OT
Yellow	Yellow	Review & Adjust Assignments
Yellow	Red	Adjust Assignments; Consider Negotiation (Schedule)
Red	Green	Decrease OT or People
Red	Yellow	Adjust Assignments; Consider Negotiation (Funding)
Red	Red	Negotiation (Funding/Schedule/Rqmts); Fire Manager

For this control charting method, the individual values of monthly project performance are plotted in their sequential order. The average of all the values is calculated and, likewise, shown. upper and lower natural process limits (UNPL and LNPL) are also shown as distinctive lines on the chart. These lines are computed to be the *six sigma* limits of the process under review. Statistical theory provides methods to calculate the UNPL and LNPL based upon the dispersion of the moving range (mR) [1 and 3]. For the application of SPC presented here, the differences between the successive monthly values for CPI^{-1} and SPI^{-1} become the data for the mR analysis.

Just as for X, the moving range is graphed. Data points are plotted in proper monthly sequence, with the computed average value of mR. As with UNPL and LNPL lines shown on the Individuals chart, the UCL and LCL are displayed as lines on the mR chart. As for UNPL and LNPL, statistical theory provides computational means for determining UCL and LCL values.

The formulas for calculating the process, or control, limits (the *six sigma* values) of the XmR charts are available in the cited text references. In our application, because the adjacent X data points are paired to form the mR data, the subgroup size (n) is said to be two. Knowing n=2, the values of the constants required by the formulas are determined from the control chart tables [1 and 3]: $d_2 = 1.128$, $D_3 = 0$, $D_4 = 3.268$.

An example of the XmR chart is illustrated by Figure 1. Note, on the Individuals chart there is another line in addition to those for the average value of X, UNPL and LNPL. This line is labeled USL (i.e., the Upper Specification Limit). The USL is not derivable from the data; it is a performance value, or constraint, that the process is not to exceed. A considerable amount of subsequent discussion concerns USL and its value in relation to "X-bar," or the average value of X, and the UNPL.

Analysis/Interpretation

The successful project manager must continually ask, "Can the project be completed if it continues performing as it has?" SPC application to CPI^{-1} and SPI^{-1} can help answer the question. Comparing the X-bar values of these EV indicators to the planned performance (i.e., to the value of 1.0) provides information about current and future performance for the project. If the value is 1.0 or less, then the project is performing well and can be expected to complete within its planned cost and schedule. If it is greater than 1.0, then there may be trouble requiring management attention.

Now we are ready to discuss the upper specification limit, or the process constraint, mentioned earlier. The USL for cost is the *cost ratio* (CR), while for schedule it is the *schedule ratio* (SR). The *cost ratio* is defined as the total funding available for the project divided by the planned value (in EV terminology, budget at completion, or BAC). The *schedule ratio* definition is the negotiated period of performance divided by the planned period of performance. The value of both ratios may exceed 1.0; the portion of the ratio in excess of 1.0 establishes the amount of management reserve available for handling project risks [7].

Comparing the appropriate X-bar value to its respective USL (i.e., CPI^{-1} to CR, and SPI^{-1} to SR) provides information to the project manager as to whether or not the project is executable if

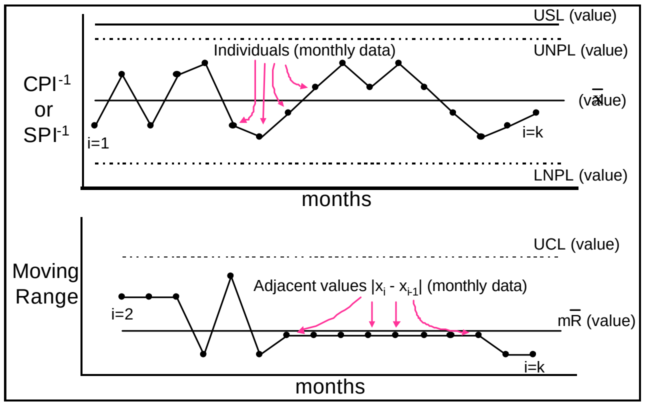


Figure 1. *XmR Example*

present performance continues (see Figure 2). This comparison is akin to the SPC analysis of process capability. In the manufacturing application, if UNPL and LNPL are within the USL and LSL, the process is said to be *capable*. If the USL, or LSL, are within the UNPL, or LNPL, calculations can be made to determine the probability of producing defective products. Corrections to the process are sought to minimize the output of defectives.

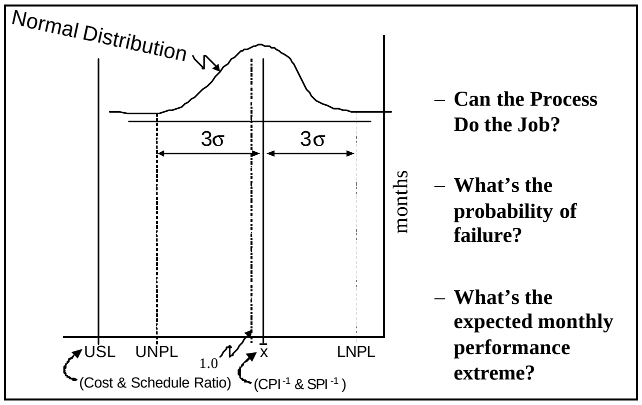
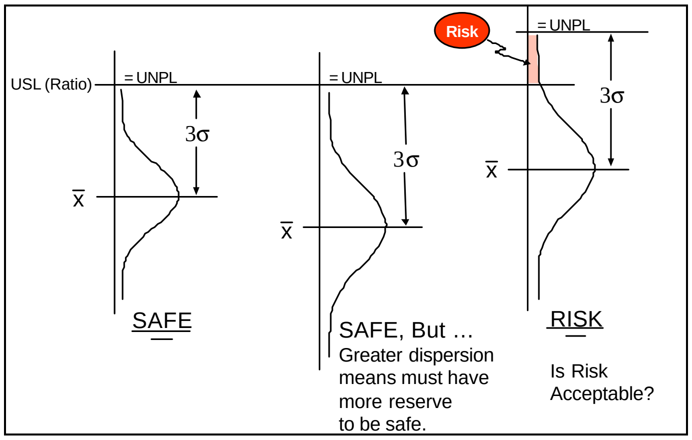

Normal Distribution
3σ
3σ
months
USL
UNPL
1.0
X̄
LNPL
(Cost & Schedule Ratio)
(CPI-1 & SPI-1)
– Can the Process Do the Job?
– What's the probability of failure?
– What's the expected monthly performance extreme?

Figure 2. *Process Capability*

The interpretation of the SPC application of process capability to software project management, and the EV indicators CPI^{-1} and SPI^{-1}, is somewhat different from the description given for manufacturing (see Figure 3). The project can have *defective* monthly performance results and still be in good shape. The process can be expected to achieve satisfactory results if the average value of CPI^{-1}, or SPI^{-1}, is less than the USL (cost or schedule ratio). Besides determining process capability, there is

Figure 3. *SPC Analysis/Interpretation*



other interesting and useful information from the SPC analysis; the probability of project failure can be determined, along with the expected monthly performance extreme. The UNPL value is the expected performance extreme. The portion of the *normal distribution* that exceeds the USL quantifies the risk of failure.

A few other observations concerning Figure 3 can be made. If UNPL equals USL, the project performance could be termed *safe* (i.e., the risk of failure is nil). If 3 sigma is large with respect to X-bar, then there is a large amount of dispersion (mR) in the monthly performance. To be *safe* with large dispersion requires a greater amount of management reserve. To be competitive, it is very advantageous to minimize the UNPL and move it towards perfection (i.e., the value 1.0). The risk, or probability, of non-performance should be minimized. By decreasing risk and, thus, management reserve, a company can decrease its bid price, and increase its chance of contract award. Also, risk can be planned for a project. The project can be managed to that amount of risk. Often that is what is done to win the bid. But, without using SPC, the bidder does not know his chance of failing. *If the bidder plans no Management Reserve, his probability of achieving the project plan is only 50 percent.* This point is easily seen from Figure 3 by lowering USL until it equals the planned performance of CPI^{-1} or SPI^{-1} (i.e. 1.0). Even if the developer has a very good process and the process variability is minimal, without planned reserve, his chance of achieving cost and schedule is only 50 percent. Fifty percent is not very good odds if the company wants to stay in business and make money.

Project Manager Use

Earlier, we alluded to the application of SPI and CPI in building a project plan. Past statistical performance can be used to build a risk strategy leading to the requirement for management reserve. The following discussion, however, will focus on project performance instead. Project performance evaluation is simple and is very similar to the description given in reference [7]. The evaluation for the SPC application to the inverse of CPI and SPI is a comparison of the average values to the planned value (1.0) and the USL (cost and schedule ratios). The evaluation criteria are shown in Table 2. If the X-bar value is less than or equal to 1.0, the project can be expected to complete as planned. If the value is greater than 1.0, but less than or equal to the USL, then the project can be expected to complete within its allocated cost and schedule. Of course, in this range of performance some of the management reserve is being consumed by execution inefficiency. Finally, if the X-bar value is greater than the USL, project failure can be expected. For ease of recognition, the status can be color coded *green*, *yellow*, or *red*. *Green* indicates the project can complete within the plan, while *yellow* means the project can be completed within the cost and schedule allocations (i.e., within the plan plus management reserve), and *red* says failure is to be expected.

Once the color status has been determined for cost and schedule performance, Table 1 can be used to determine needed management action. Getting to corrective action in Table 1 is a fairly simple matter. *Red* performance status requires management attention. If the project is not performing well, something must be done; corrective action is needed. Eyebrows will raise when *yellow* occurs, but a more in-depth look should be made before any correction is made. Evaluating the performance of CPI^{-1} and SPI^{-1} using statistical methods leads to appropriate actions.

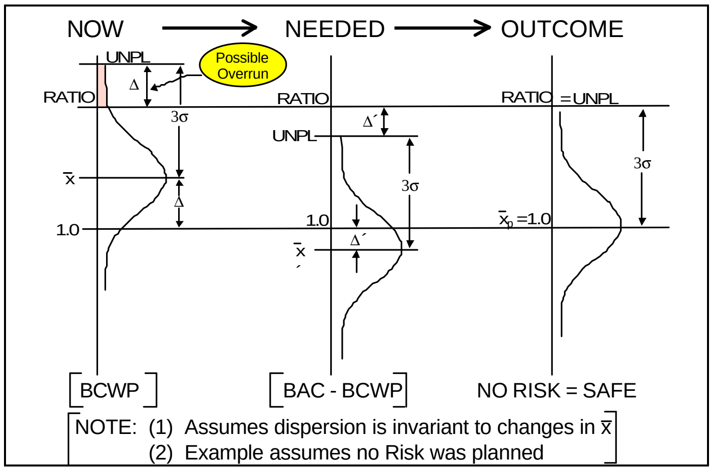
Beyond understanding what to do to bring the project back in line, software project managers need to know the extent of correction necessary. Specific areas in which SPC can be used to quantify correction are adjustments to overtime and staffing, and funding and schedule negotiations. If the project manager has resigned himself to negotiation for correcting the project's ills, the amount of overrun of funding or schedule with respect to the customer agreement can be easily determined. Overrun is found by simply multiplying the difference between the X-bar value and its respective USL times the BAC for funding, or planned period of performance for schedule. The quantity calculated is the amount needed to raise the probability of meeting these revised performance requirements to 50 percent, thus, the minimum amount to be pursued in the negotiation. Additional funding or schedule must be added to the minimum value if the desire is to provide some amount of management reserve for the remaining portion of the project. In general, these negotiations are not easy. The developer has a tendency to understate real needs of the project. By not coming to terms with actual performance, he settles for less than the full amount needed to successfully deliver the product, and ends up having to negotiate again. Generally, the second negotiation is considerably more unpleasant than the first. Unless there is no other source for the products or services, the software organization will not likely be awarded another contract by the customer.

The corrective action concept for adjusting overtime and staffing is illustrated by Figure 4. If efficiency has been poor, something must be done for the remainder of the project to raise

Table 2. *Evaluation criteria for X-bar cost and X-bar schedule*

$\bar{x} \le 1$	GREEN 🟩	Project can be completed as planned
$1 < \bar{x} \le$ USL	YELLOW 🟨	Project Manager and employees get to keep their jobs
$\bar{x} >$ USL	RED 🟥	A bad situation for those involved

Figure 4. *Project Manager Application*



the efficiency so the project completes within allocated cost and schedule. The possible overrun is symbolized in the figure by Δ (delta), and the amount of correction needed for the remainder of the project is symbolized by Δ' (delta/prime). The Δ condition has existed over a portion of the project (i.e., BCWP, the earned value for the tasks completed or in work). The Δ' condition will need to occur over the project remainder (BAC – BCWP) to bring the overall performance to the desired state. Figure 4 illustrates the theory; however, it implies that poor performance can be corrected to the ideal performance condition. Please disregard the implication. *Corrective action is not free*. It is achieved at the expense of cost or schedule. You may be able to correct schedule, but it will be at the increase of cost. Unless true efficiency gains are achieved, management reserve is expended for the corrections.

The method for calculating the X-bar value needed for achieving project cost and schedule requirements is simple, and based upon the same concept as the *To Complete* indices used in EV Management [6]. The calculation method is depicted in Table 3. The result of the correction X-bar (', X-bar/prime) is used in the overtime and staffing equations shown in Table 4. The strategy for recovery can be banded by recalculating X-bar/prime, using total funding available in place of BAC. Recalculating the overtime and staffing adjustments with the extreme X-bar/prime value determines the minimum adjustments the project can make and still achieve its negotiated cost and schedule.

Project Changes

Legitimate questions regarding this application of SPC are "What happens when the project is replanned?," and "Can the data from the XmR chart prior to the replan be used with the data obtained afterwards?" Obviously, the answers come from the changes caused by the replan to the software project's EV system. If (think of this as a manufacturing example) the only change is in the quantity of products, it follows, there will be no adjustments made to the work breakdown structure (WBS) or the earned value of the tasks. For this type of replan, the old data can be used with the new. If, however, the WBS or the task values are changed by the replan, then the remainder of the project must be treated as though it is a new start.

Table 3. *Performance Correction Index*

- For Schedule or Cost "curve shifting":

$\Delta = \bar{x} - 1.0$ [shift away from plan]

$\Delta' = \Delta \bullet \dfrac{\text{BCWP}}{\text{BAC-BCWP}}$ [performance correction to achieve plan]

$\bar{x}' = 1.0 - \Delta'$ [required performance index for remainder of project]

Table 4. *Adjusting Overtime and Staffing*

- Schedule Recovery (Reserve Funding is possibly used)

$E_{SR} = (1 / \bar{x}'_{SCHED}) \bullet E_P$
where E_P = planned number of employees

$OT_{SR} = (1 / \bar{x}'_{SCHED}) \bullet (1 + OT_P) - 1$
where OT_P = planned overtime rate

- Cost Recovery (Schedule Reserve is used)

$E_{CR} = (\bar{x}'_{COST}) \bullet E_P$

$OT_{CR} = (\bar{x}'_{COST}) \bullet (1 + OT_P) - 1$

- Band the Recovery Strategy
  - Substitute Total Funding Available for BAC in the Δ' calculation

Application

In our organization, projects have been managed using the cumulative values of SPI and CPI for some time. Actual project monthly data is plentiful and readily available to create and test the SPC application described. One project's data is exhibited in Figures 5 and 6, the XmR charts for CPI^{-1} and SPI^{-1}, respectively. Although not shown, the histograms prepared from the performance results for both CPI^{-1} and SPI^{-1} approximate *normal* distributions. Although there are SPC experts who will argue that not having a normal distribution of the data does not invalidate the use of Control Charts, some confidence is created in this application when it is seen that the distributions appear *normal*.

To begin the SPC analysis, refer first to Figure 5 for the discussion of the CPI^{-1} Control Charts. As can be seen, the process can be said to be reasonably well controlled; the average value of mR is fairly small (0.2652). Even so, the variance is large enough to place the computed value of UNPL above the *cost ratio* (USL). Recalling the earlier analysis/interpretation discussion, when UNPL is greater than USL, there is a computable probability that

Figure 5. *Software Development Project CPI^{-1} Data*

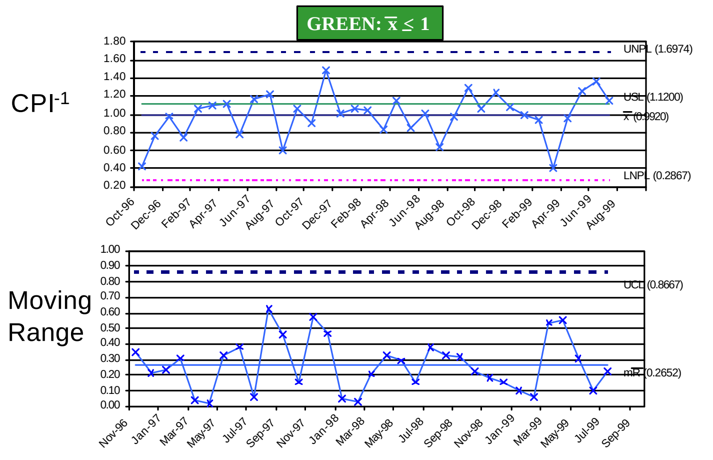
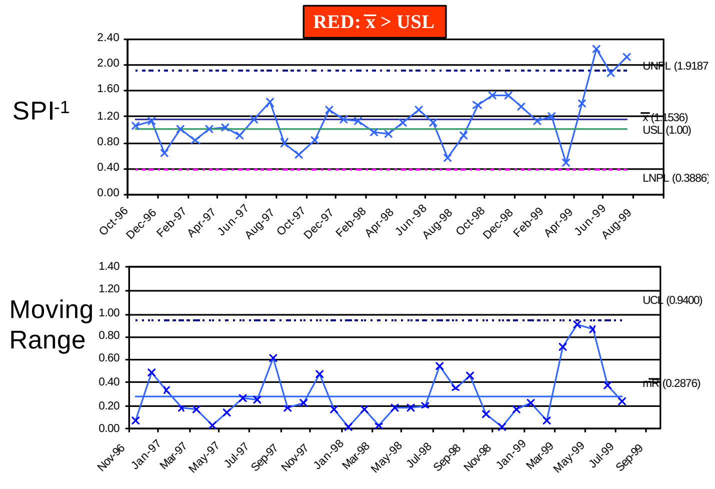


Figure 6. *Software Development Project SPI^{-1} Data*



the project will exceed its allocated cost. Performing the mathematics, and using the normal distribution table [8], the probability of overrunning the *cost ratio* is determined to be 29.3 percent. Observed directly from the CPI^{-1} graph, worst expected monthly performance (UNPL) is 1.694. The average value of CPI^{-1} (X-bar) is seen to be less than the *cost ratio*, so the process can be said to be *capable*. Likewise, X-bar is less than 1.0 and, thus, the status indicator color is *green*. The project manager can expect the project to be completed within its planned cost. With very good numbers, the project manager does not have much worry here.

Let us shift our attention to Figure 6. The control of schedule is not as good as for cost. The average value of mR (0.2876) is somewhat larger, and, correspondingly, the value for UNPL (1.9187) is computed to be larger than for the CPI^{-1} Control Charts. The average value of SPI^{-1} is greater than one, and greater than USL, a *red* status condition. Here is something to worry about—not meeting the customer's schedule. Using the *schedule ratio* and the average value of SPI^{-1}, and the normal distribution table [8], the probability of failure is found to be 72.7 percent. This is definitely not good; the condition needs management action. Before the recommended action is described, a few observations can be made. *This project planned no management reserve for schedule (USL=1.0).* The only way this project can correct its poor schedule performance, other than having a miraculous improvement in schedule efficiency, is through the use of the cost reserve. It is always better to have reserve in both cost and schedule.

Returning to Figure 6, focus on the last three data points (May, June, and July '99) of the SPI^{-1} graph. They are especially interesting. These points, which hover around UNPL, are anomalous. The probability of these data points occurring is infinitesimal. Their existence is virtually impossible. From SPC theory, this behavior leads the analysis to seek an assignable cause [1 and 3], some influence outside of the system. Reviewing the same months on the CPI^{-1} graph, the data appears to be reasonable. From EV analysis, we know, if CPI performance is good and SPI is poor, then the project is likely suffering from a manpower shortage. Actually, the project had several software engineers hired away (by a single employer) in those months and, correspondingly, the manager was unable to hire a sufficient number of replacement employees. Also, we found there were other significant contributors to the poor schedule performance: the impact of the huge Oklahoma City tornado that affected many of our people from May through July; unplanned Air Force-driven training; and, a computer security alert that required the staff's attention. In proceeding with the analysis, a decision should be made as to whether to include the data attributable to an assignable cause. Because our purpose is merely to demonstrate the calculations, we have chosen to retain the data.

From Table 1, management's correction strategy is *increase overtime or staffing.* Next, the schedule recovery formulas of Table 4 are used to quantify the necessary increases. The formulas require the value of the schedule correction index, X-bar /prime. Knowing the values for X-bar (1.1536), and BCWP/ BAC (.617), the completed portion of the project, X-bar/prime is computed to be 0.7526 from the equation given in Table 3. Having the value for X-bar/prime, we can determine if the schedule correction can be accomplished by simply increasing overtime. For the completed portion of the project, the effective overtime rate is 7 percent. The rate required for the remainder of the project is determined to be nearly 42 percent. Working employees at 42 percent overtime for very long is not advisable. Doing so will likely increase the loss of employees and worsen the problem. More employees are needed. Dividing the effective number (59) of employees for the completed portion of the project by X-bar/prime (0.7526) yields 78, the number of employees needed for the remainder of the project. It is interesting to note that the project manager, in his effort to determine the corrective action, reported the same number of employees after spending two days manipulating a commercial project scheduler. Certainly, this single correspondence with a commercial project scheduler is not enough evidence to say the methods described here will always provide good management information, but it does greatly increase the level of confidence. Also, the 10 minutes we spent making the estimate compares very favorably to the two days needed by the project manager to come to the same result.

Summary

The concepts of statistical process control, specifically control charts, have been discussed with respect to software project management of cost and schedule. SPC Control Charts are shown to have a practical application to the EV indicators, Cost and Schedule Performance Indices. SPC can be easily utilized by software projects using Earned Value Management. Furthermore, example results discussed indicate this application of SPC can be useful. It appears that coupling SPC Control Charts to EV has the potential to be extremely powerful. The methods and techniques described can be used for:

- Quantifying Problems.
- Process Capability.
- Probability of Failure.
- Worst Expected Monthly Performance.
- Developing Recovery Strategies.
- Overtime.
- Staffing.
- Negotiation Values.
- Project Planning.
- Quantifying Risk.

Final Thoughts

Beyond project planning and control application described in this article, implicit in SPC is *process improvement.* An example of *assignable cause* was provided in the discussion, but just as important is the improvement of *common cause* entities [1, 2]. Improvement to common causes reduces variability of the process and, thus, risk. For the software development organization, the reduction of risk leads to decreasing the requirement for management reserve, thereby making the organization more competitive. And for the customer, reduced variability in the software developer's process means lower prices and greater probability of achieving cost and schedule. Everyone wins—it is powerful stuff.

▸▸ *This article is continued on page 28.* ▸▸

What We Have Learned

by Lawrence H. Putnam and Ware Myers
Quantitative Software Management Inc.

In all the changes in the software field in the last quarter century, one solid element has been the SEI core metrics: functionality (usually expressed as a measure of size), schedule time, effort (convertible to cost), and defect rate. From these four metrics, we derive a fifth, productivity. These five are related to each other. From these relationships, we derive a number of lessons, which it is the business of this article to identify.

Lesson 1. Conventional productivity is deceptive.

Productivity in most manufacturing operations is *linear;* that is, when a cobbler worked longer hours, he produced more shoes. When he hired an assistant, he doubled production. Naturally, this frame of mind carried over into software development. Software managers adopted the linear expression, source lines of code (SLOC) per person-month, as their gauge of productivity.

This is not the correct frame of mind. When we examined the actual data, source lines of code and person-months on thousands of completed projects, we found two facts to the contrary:

- It takes more effort per source line of code to build larger systems than to build smaller systems. Another way of putting this is to say if effort per SLOC were the same at all system sizes, the relationship would be *linear.* Since effort per SLOC is not constant at all sizes, the relationship is *nonlinear.*
- Moreover, at any one system size, the effort required to build it differs widely. For example, in the project data we have collected at a size of 100,000 SLOC, effort ranges from a low of 10 person-months to a high of 2500 person-months. The extent of the effort is equally great at other system sizes.

These facts should not be unexpected. Large systems generally are more complex than small systems. The relationships between their parts are more intricate. Larger teams of developers complicate their interactions, as compared to small teams, and so on. Similarly, with respect to the second fact, there are great differences in system complexity between developments in different application areas. Experience tells us, for instance, that real-time systems take far more work than business systems of the same size in lines of code.

This nonlinearity of the relationship between SLOC and effort means that the conventional productivity a software organization derives from its experience on one size or application type is not a reliable guide to estimating the next system of a different size or application type.

Let us bring in a second core variable, *time*. Again, we see the same nonlinearity.

- Development time per SLOC increases with system size.
- Development time varies widely at each system size. For instance, at about 10,000 SLOC, development time ranges from about two months to 80 months.

We learned that measuring software productivity in the conventional way, SLOC/person-month, is deceptive. Software productivity is more complicated than that.

- Productivity is affected, not only by effort, but also by *time.*
- The relationship of the three metrics—size, effort, time—is *nonlinear.*

The true relationship is a *nonlinear* version of SLOC/(person-months^{a} × calendar months^{b}) The exponents, *a* and *b*, represent nonlinearity. We call the result *process productivity.*

Lesson 2. There is a minimum development time.

There is always pressure on development organizations to get the system out faster. This is quite understandable. We live in a competitive society. Faster is better. As we examined the recorded metrics of hundreds of systems over these 25 years, however, we came upon an interesting reality—no one had ever completed a system in less than a certain *minimum development time.*

We came upon this fact in the course of the studies that led to Lesson 1. There were no data points below a certain value of schedule time and they were mighty sparse just above that value. There are no data points for a simple reason. No one has been able to do a system in that short a time.

Of course, one has to define what the system is. It is a specified collection of requirements or features at a quality level. If you sacrifice some of the features, you have less work and you can proportionally reduce the minimum time. If you sacrifice quality, for instance, by abbreviating system test, you can ship a little sooner. To complete the originally intended system, however, takes a certain minimum amount of time. You cannot beat that minimum time simply by adding more people.

Knowing what the minimum development time of the prescribed system is, based on hard metrics, can be a great comfort when you are standing before the decision-makers. Contrariwise, when the decision-makers are contemplating a set of bids, they can use that same knowledge to throw out the bidders who do not understand metrics. The low bidder is not always a bargain.

Lesson 3. You can trade off time and effort.

The minimum development time sets a lower limit. At this limit, effort (or staff, or cost), and defects are at a maximum. You may not be happy about operating at minimum time. After all, cost and quality are important values, too. By extending development time, you can reduce effort and defects. The upper time limit is a matter of judgment, but it is around 130 percent of the minimum time. Beyond that point, further reductions in effort and defects are small. Within that time range, you can balance time, effort, and defects to fit your business pressures.

Lesson 3a. Small is better.

Other things being equal, a small staff is more efficient than a large staff. It is more efficient because there are fewer interfaces between project members. Staff spends less time communicating, but keep better informed.

Our metric observations have established the reality of small being better all along, but a couple of years ago we obtained data on 491 medium-sized projects. The two- to five-person teams completed projects of comparable size with about one-third of the effort of the seven- to 14-person teams.

Lesson 3b. Fewer defects are better.

There is also a time-quality tradeoff. When you extend your development time beyond the minimum, you score fewer defects. Sophisticated users are noticing that the first product to market generally tries their patience.

Lesson 4. You can live with uncertainty.

When we collected data on hundreds of projects[1], the uncertainties in any core metric were balanced by offsetting uncertainties in other metrics. As an individual, you are dealing with one project at a time, yet the data you must work with is uncertain. For example, in estimating time and effort, you start with values for system size and your project team's process productivity. At the time you have to bid, both the system size and the process productivity of the team that you will assemble for that project are uncertain. By *uncertain* we mean that you cannot pinpoint the value. You estimate the size, for instance, to be 77,000 SLOC, plus or minus 11,500 SLOC.

The method of statistical simulation, summarized in the sidebar, enables you to cope with these uncertainties. Briefly, this method enables you to translate the uncertain values of your input values into the probability that you can successfully complete your project at the values of schedule and effort that you select. You have to select values, of course, within the range of what is possible.

Without the benefit of this kind of analysis, management is often tempted to go with the low bid. Granted, it is important to win contracts. It is also important to complete projects successfully and make a little money while doing so. It is also important to stay in business. This type of analysis gives you the opportunity to gauge your chances.

Lesson 5. You can control a project under way.

Our study of completed projects reveals that key ongoing metrics, such as the amount of effort, the number of staff, functions completed, or defects detected, fall along a Rayleigh curve, as diagrammed in Figure 2. Great! All you have to do is figure out a way to project the likely occurrence of each of these variables. (Hint: The methods are available.) Moreover, since these methods are statistical, you can project control bands, also shown on the figure.

That provides the control side for *statistical control.* The other side is to plot actuals week by week on the control diagram. If the actuals are falling within the inner control band, work is progressing as expected. If the actuals begin to veer out of the control band, something has gone wrong. You have early notice of trouble.

Before we leave this lesson, let us note a frequent exception to the Rayleigh pattern. It is staffing. If management staffed a project in accordance with the needs of the project, staff would build

Deal with Uncertainty Statistically

Statistical simulation, summarized in Figure 1, provides the means of dealing with the uncertainties of your input values.

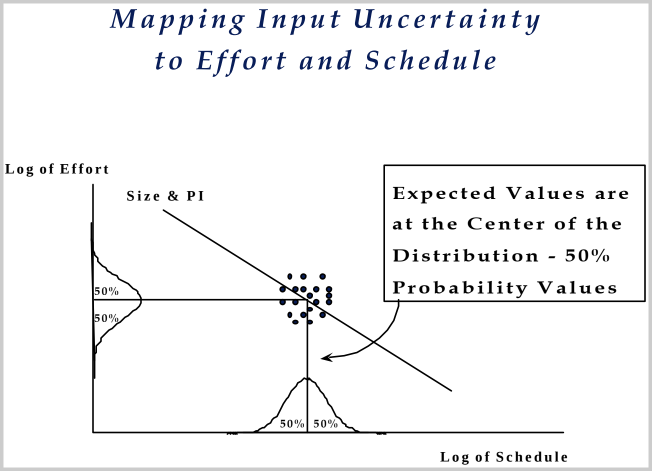

Mapping Input Uncertainty
to Effort and Schedule
Log of Effort
Size & PI
Expected Values are at the Center of the Distribution - 50% Probability Values
50%
50%
50%
50%
Log of Schedule

Figure 1. *Computing your project estimate a thousand times provides a thousand possible answers, symbolized by the black dots. Projecting the thousand data points to the two axes enables you to find the probability that any one answer will work out.*

Looking at Figure 1 in terms of seven successive concepts makes it easier to understand. We have to admit that the figure can be a bit awe-inspiring at first glance, so do not glance at it. Study it one concept at a time.

1. The diagonal line labeled "Size and PI" (process productivity) symbolizes the location of your possible operating points somewhere between the minimum development time (a bit to the left of the circle of data points) and 130 percent of it (to the right of the data points).
2. Let us say the center of the circle of dots is the operating point you have chosen, that is, a particular length of schedule and amount of effort.
3. You then have a statistical program on your computer calculate schedule and effort a thousand times, leading to a thousand values of effort and schedule, symbolized by the circle of dots.
4. For each computation, the statistical program selects input values from a statistical distribution range around the values of size and process productivity you are using. Since the input values of size and process productivity vary, the output values of time and effort also vary. This variation is symbolized by the distribution of dots in the output circle. If we were to show a thousand output data points, we would find that they follow a probability distribution, heavy toward the center of the circle of dots, light toward the circumference.
5. Next we project the data points to the two axes. The most likely values of effort and time are the ones at the peak of each curve. In terms of probability, the chance is 50 percent that the project can be accomplished at those values.
6. Selection of a longer schedule (to the right on the schedule curve) increases the probability of project success, while at the same time reducing the effort needed within the limits that the two curves provide, of course.
7. Selection of a shorter schedule, however, reduces the probability of project success while increasing the effort needed.

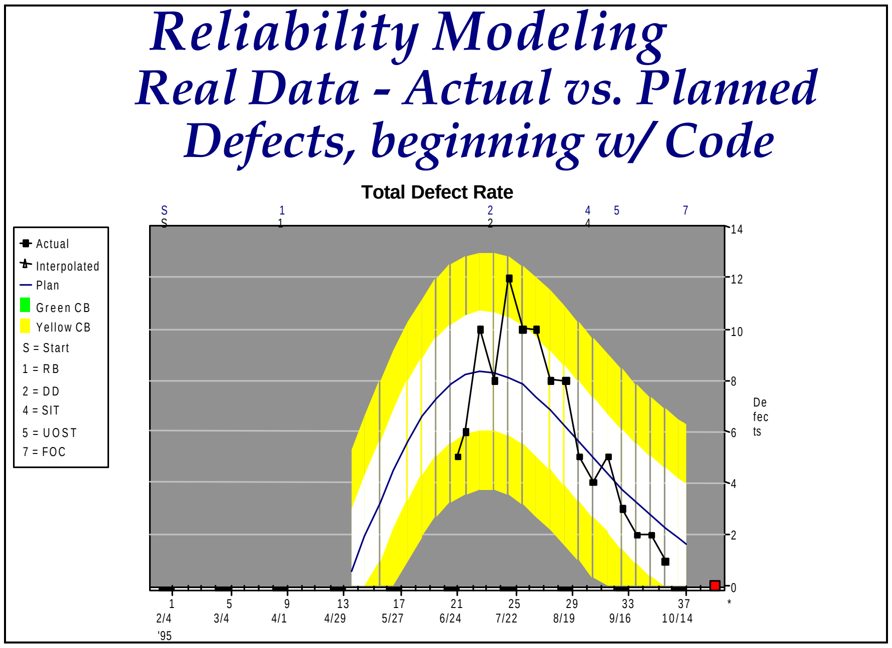


Figure 2. *The solid curve is the projected defect rate. The little squares are the number of defects discovered each week. The band above and below the solid curve represents the allowable tolerance around the defect-rate curve. In week 24 the number of defects peaked, but the project manager got right on it!*

up over a period of time as the small initial staff sorted out tasks to occupy more and more people. At some point as tasks in work reached a peak, so would staff. As the project wound down, so would staff.

Rayleigh staffing, however, takes continuous management attention, moving people from declining to growing projects. Management often prefers level staffing, assigning full staff at project initiation, and maintaining it until completion. In the all-too-common disaster scenario where risks and defects are left until integration or system test, full staff and more are needed right up to the release bell. To the extent that considerations other than task needs dictate the staffing pattern, it may not follow the Rayleigh curve.

Lesson 6. You can replan a project midway.

In spite of the best-laid plans, projects often go astray. Moreover, in software development, the plans may not be laid all that well. If we are accumulating metrics as we go along, we can use them to project a new schedule and effort to completion.

The new plan is likely to extend the schedule and perhaps to call for more staff than the budget can afford. In that event, you can cut features to meet the schedule required or the budget available. Metrics, as such, cannot tell you what to do. It can provide you with the informational means to do what your circumstances require of you.

Lesson 7. You can monitor process improvement.

With the introduction of the Unified Process about a year ago, many organizations have more management attention focused on process improvement than ever before. Management attention is a scarce resource; managers have a score of urgent matters clamoring for attention. It helps to focus their attention by having a metric cross the desk periodically. In the realm of process improvement, such a metric is process productivity.

We derive process productivity by formula from the size, time, and effort metrics of each project. Hence, it is a definite number. In an organization with many projects, it is also a frequent number. It was originally usedfor project estimation, but you can use it to pursue other goals as well:

- If the average process productivity of projects completed this year exceeds that of last year, you are making progress. Moreover, the difference between the two numbers indicates your rate of progress. You can be satisfied, or you can be dissatisfied and take action.
- The average process productivity of the organizations reporting data to our database has improved for the last two decades. You can project a comparable improvement rate and plan the actions it will take to achieve it.
- You can compare your level of process productivity with other divisions within the same corporate structure or with industry-wide averages we maintain.
- You can evaluate the process productivity level of subcontractors on whom you have some leverage, such as a $1 million contract to bestow.

One use of process productivity we urge you not to employ is evaluating a particular project, its project manager, or its staff for personnel-type purposes. Software metrics are too important for all these broader management purposes to risk muddying them to assess individuals.

Lesson 8. You can profit from experience.

No individual has more usable experience than he or she can cram into that famous little black book. Often it is as little as two or three projects. Work done long ago in a chaotic process on a prehistoric operating environment may no longer be relevant. Yet the agency or corporation in which you labor may have completed scores of projects in the last few years, if only you could get your hands on that experience.

Obviously, you do not have time to dig through the written records in hundreds of file drawers, perhaps scattered around the globe, perhaps in dusty dead-record warehouses. The answer in the personal computer age is a metrics database accessible from your desk or laptop.

The general answer, the metrics database, is easy to give. Making it work is a little more difficult. Your organization has to decide what metrics to store, but start with size, time, effort, and defects—the SEI core metrics. It has to define those metrics so that they mean pretty much the same thing from project to project and from location to location. It has to institute the discipline that it takes to collect data from not-too-enthusiastic sources, at least in the beginning. In time, everyone will glory in the ready availability of good metrics.

In other words, we have learned that metrics has day-to-day value only if they can be readily accessed.

Lesson 9. You can master plan.

There is a level of activity above the project. With our strong tendency to focus on the problems of the project, we overlook the fact that most organizations have a number of projects under way at the same time. They are not all at the same process phase. Their need for resources differs.

If you have common metrics, if the metrics for each project is in your computer, you can master plan the allocation of resources over time to each project. You can prioritize your projects to match your resources, or with the advance notice that the master plan provides, you can build up your resources to meet coming needs. In reverse, if the master plan forecasts a lull six months hence, you can slow down hiring or bid new work a bit more aggressively.

As component-based development gets more play, the source of these components comes into focus. That source, wherever it is, is outside the usual project emphasis. You may have a supraproject group developing common architecture, standardizing interfaces, and developing components for a whole range of projects. You may be obtaining components from vendors. You may be charging one of your own projects with generalizing a component it needs for its own application for broader use. You need metrics on the master-plan level to facilitate component-based development.

Wrapping It All Up

Underlying these nine lessons is the software equation. We will not go into its details here. What we learned from it is that schedule time and effort-cost play a simultaneous role in software development. The two are irretrievably linked. You cannot have one without the other. Trying to play them separately is what has led to a lot of the trouble that has beset the field.

"What we learned . . . is that schedule time and effort-cost play a simultaneous role in software development. The two are irretrievably linked."

The software equation is also at the heart of the idea of calibration. To know what you have to do to get to where you want to go, you first have to know where you are. That is what we mean by calibration. The software equation provides that means.

For example, from the software equation we derive process productivity. Knowing the productivity of our process is a key ingredient of planning and estimating. During a project we can find out if we are getting the process productivity we originally estimated. If we are, that is good. If we are getting something less, we can replan the remaining work before it is too late. Finally, process productivity provides the basis for evaluating productivity between projects and over time. It is the beacon light for your process improvement activity. Process improvement is what keeps you in business in these tumultuous times.

Further Information

The Web site qsm.com lists about 40 articles, explaining the lessons of this article at greater length. In addition, Putnam and Myers are authors of three books:

Measures for Excellence: Reliable Software on Time, Within Budget, Prentice-Hall, 1992;

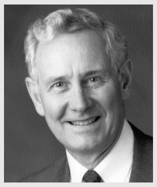
Industrial Strength Software: Effective Management Using Measurement, IEEE Computer Society Press, 1997;

Executive Briefing: Controlling Software Development, IEEE Computer Society Press, 1996.

Note

1. Now the number of projects is over 5,000.

About the Authors



A graduate of West Point, **Lawrence H. Putnam** spent 25 years on active duty, including tours in the Office of the Director of Management Information Systems and the Assistant Secretary of the Army, Financial Management. There he viewed the problems of software development from a top-management perspective. In 1978 he founded Quantitative Software Management Inc. in McLean, Va., and continues today as its president.

Quantitative Software Management Inc.
2000 Corporate Ridge, Suite 900
McLean, Va. 22102
Voice: 703-790-0055
Fax 703-749-3795
E-mail: Larry_Putnam_Sr@qsm.com

Ware Myers graduated from Case Institute of Technology, and earned a master's degree from the University of Southern California. As a contributing editor of *Computer* magazine, he helped Putnam in 1981 with his first tutorial book for the IEEE Computer Society, the start of a long writing collaboration.

1271 North College Ave.
Claremont, Calif. 91711
Voice: 909-621-7082
Fax: 909-948-8613
E-mail: myersware@cs.com

Web Addition

The V Model

The author of this article, which may be found in the online edition of CrossTalk, is the technical project officer for the Data Exchange Agreement for Software Technology between the United States and Germany. It was in this capacity that he became aware of the German software standards, known as the V Model, for the German Federal Armed Forces. The standards are published in three volumes and can be tailored to fit officially sponsored work. In this Web Addition, the author introduces these standards to give readers a flavor for them and to encourage learning more about software standards used by a political and military ally.

by Morton Hirshberg
Army Research Laboratory

Proven Techniques for Efficiently Generating and Testing Software

by Keith R. Wegner
Northrop Grumman Corp.

Generating reliable, error-free software on time and within budget is becoming ever more important as competition increases and procurement budgets shrink. In response, software engineers are continuously striving to develop and implement processes that more efficiently bridge the gap from system analysis and design to embedded systems. This paper presents a proven process that uses advanced, commercially available, MathCAD® and MATLAB® tools to design, develop and test optimal, error-free embedded software.

The MATLAB programming language is quickly and unobtrusively becoming a primary prototype and analysis tool at Northrop Grumman Electronic Sensors & Systems Sector (ESSS) as well as at other engineering entities the world over. Its value to both systems and software engineers has been well demonstrated on many programs at ESSS, including Comanche, and it is especially well-suited to effectively support the corporate Integrated Product Team (IPT) concept. Engineers desiring to be technologically competent and efficient may soon find a solid foundation in MATLAB to be almost indispensable. Its application is becoming pervasive within systems engineering groups, and it is rapidly spreading to other disciplines, such as software, rapid prototyping, and financial analysis. On a similar note, MathCAD is routinely used in the systems engineering disciplines to perform detailed algorithm analysis and development. Although not as well-suited for large, elaborate simulations as MATLAB, this tool excels in performing symbolic manipulations and in developing and documenting detailed mathematical derivations. It is especially renowned for its representation of equations using succinct, precise mathematical notation. Systems engineers are routinely using both MathCAD and MATLAB to develop and capture their design documentation in Mode (MDD) and Function (FDD) Description Documents. Thus, software engineers with a working knowledge of MathCAD and MATLAB will be well positioned to much more efficiently develop software products by bridging the gap from systems MDDs and FDDs to embedded software. These two tools are especially powerful when applied to processes that are inherently mathematical, such as stochastic processes, image and signal processing, inertial navigation, etc.

This paper describes informal techniques developed and successfully applied by the author, within the governing software process, on the Comanche program at ESSS to produce robust, optimal software directly from a systems FDD captured in a MATLAB simulation. In this process, MathCAD was first used to quickly generate the algebraic equivalent of involved Kalman filtering matrix equations expressed in MATLAB. These expanded representations were then coded and tested in MATLAB before being translated into the target Ada programming language using available text-editing tools. MATLAB was extensively used to rigorously unit test the deliverable Ada software product. This deceptively simple process allowed the author to design and develop large, efficient amounts of code in a very short time. Furthermore, the final software was found to be virtually error free, making the successful unit testing of code so developed almost a foregone conclusion.

Background

The Comanche program's Target Acquisition System Software (TASS) contains a number of Computer Software Configuration Items (CSCIs), one of which is the Target Threat Manager (TTM) CSCI. This CSCI maintains tracks of various fidelities on numerous targets detected by assorted sensors from different sources. As such, it contains an association component that attempts to correlate new detections with established track files maintained in a target threat database (TTDB). A number of Kalman filtering algorithms implemented in the Target State Estimator (TSE) component initialize and update entries in the TTDB. These algorithms provide for 3, 6 or 9 states, with independent or partially correlated measurements, depending on the operational mode and the target type. Video TV and FLIR sensors, for example, can be scanned or operated in a 30Hz stare mode, resulting in different measurement dependencies. After performing detailed timing analyses on the target processor hardware, it was determined that the Kalman algorithms should, for example, be optimized by eliminating loops to minimize floating point operations (FLOPs). To this end, an efficient mechanism for optimizing the many mathematical operations inherent in Kalman filtering and coordinate transformation operations was developed.

Initially, the TTM systems engineering design was captured using traditional word processing and drawing tools, such as Microsoft PowerPoint and Microsoft Word. The supporting system simulations were developed in the Ada and C programming languages. However, due to the author's influence, the advantages of MATLAB over Ada and C for developing and maintaining the systems simulations were quickly realized, and a TTM systems simulation was subsequently developed entirely in the MATLAB programming language. It was soon quite obvious that modules of the well-documented MATLAB simulation code, supplemented by other descriptive documents as necessary, were, in effect, equivalent to an FDD. At this point, the author developed the techniques described in this paper to generate optimal, virtually error-free, Ada software directly from the MATLAB simulation/FDD.

Process Description

The overall process, as depicted in Figure 1, is described as follows: Given what in MATLAB is a concise, mathematical expression, use the symbolic capabilities of MathCAD to algebraically expand or "unfold" the operation. This is typically done, for example, to eliminate inefficient looping operations in multiple matrix multiplications or inversions. The expanded MathCAD results are coded in MATLAB. MATLAB is used instead of other higher-order languages because it is interpretative

MathCad is a registered trademark of MathSoft Inc.
MATLAB is a registered trademark of The MathWorks Inc.

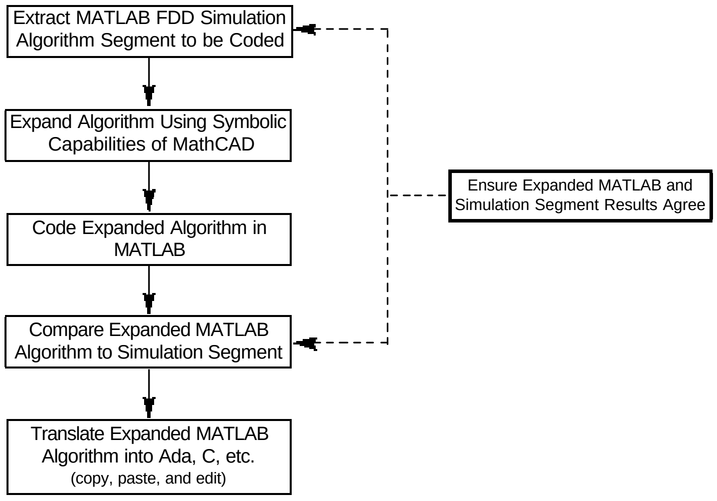

Extract MATLAB FDD Simulation Algorithm Segment to be Coded
Expand Algorithm Using Symbolic Capabilities of MathCAD
Code Expanded Algorithm in MATLAB
Compare Expanded MATLAB Algorithm to Simulation Segment
Translate Expanded MATLAB Algorithm into Ada, C, etc. (copy, paste, and edit)
Ensure Expanded MATLAB and Simulation Segment Results Agree

Figure 1. *An Efficient Process Generates Optimal Code*

and results can be quickly confirmed using well-established, embedded routines, and diagnostic techniques are easily applied. Once the expanded algorithm is available in MATLAB, its results are verified by comparison with those of the original, concise code segment from the FDD. Test data, vectors, matrices, etc. are easily generated in MATLAB to expedite this process. When both versions of the algorithm generate equivalent results, thereby validating the expanded algorithm, traditional code editing tools are used to copy the expanded algorithm and paste it into an appropriate file for the target programming language. For the Comanche TTM application, Ada was the target language, but any other higher-order language could have been used. That file is edited to conform to conventions of the target language as well as applicable program coding standards. This final step is the key. Very little new code is generated and perturbations to and modifications of the expanded and already tested MATLAB code are minimized. The bulk of the expanded algorithm is simply tailored to conform to the semantic requirements of the target language. Generally, this involves little more than global find-and-replace operations. For example, if the target language is Ada, the MATLAB "=" is replaced with Ada's "=". Since MATLAB is interpretative, any implicit objects that it uses must be explicitly defined in the target language. Although this overall process may at first seem to be fairly elaborate and perhaps even unnecessary, an attempt to manually optimize even mildly complex 6-by-6, let alone 9-by-9, matrix and vector operations will quickly demonstrate the limitations of pencil and paper. The described process has been found to generate very reliable code, equivalent to the original algorithm segment as specified in the MATLAB FDD.

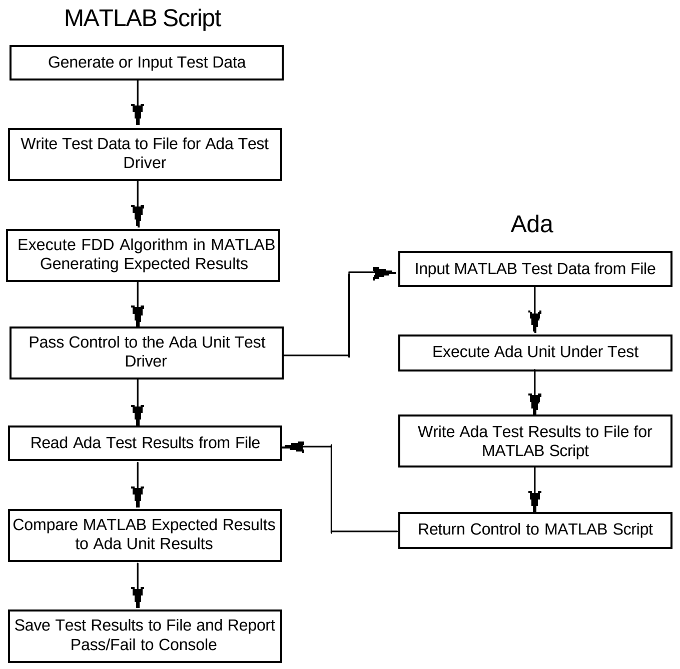
The advantages do not stop there. When it comes to the unit testing of elements generated by following the above process, MATLAB again excels. The author has developed and successfully applied an automated MATLAB-based process, as outlined in Figure 2, to generate repeatable, self-documenting unit tests. To summarize, the unit test process employs a MATLAB script file that executes the succinct code segment copied from the MATLAB FDD to generate expected results. The required unit test data is either generated internally by the MATLAB script or read from a previously prepared input file. That script writes the equivalent unit test data to a file to be input by the Ada unit test driver software. The MATLAB script then calls the executable Ada test driver that reads in the test data and exercises the element under test. When finished, the Ada test driver writes the results generated by the Ada element under test to a file for input by the MATLAB script. The MATLAB script resumes execution, reads in the data from the Ada test driver, and compares that data to its previously generated expected results. Finally, it saves test results to a unit test data file for permanent documentation and reports its findings to the console. The entire sequence is repeatable and can easily be tailored to perform multiple iterations over different random or predefined sets of test data.

Example

To demonstrate the techniques without overwhelming the reader or exceeding the physical capacity of this paper, an example of a portion of a simple 6-by-6 covariance matrix extrapolation extracted from a typical Kalman filtering application is demonstrated. For readability, the addition of the state noise covariance matrix is omitted. The equation to be developed, in any dimension, is $\mathbf{P} = \Phi \mathbf{P} \Phi^T$ where Φ is the state transition matrix, Φ^T is its transpose, **dt** is the sampling interval and **P** the symmetric covariance matrix. This is almost exactly how it appears in the MATLAB simulation. As shown below, the symbolic capabilities of MathCAD are used to generate the algebraic equivalent of this equation. The inherent symmetry of P is exploited in both the MathCAD derivation and the subsequent translation into MATLAB.

This last matrix result, which is symmetric, is now expressed in a MATLAB m-file that compares the expanded solution with the simple $\Phi \mathbf{P} \Phi^T$ matrix product. Although at first this may appear to be a somewhat daunting task, close examination reveals significant symmetry and repetition, except for indices, that greatly simplifies the MathCAD-to-MATLAB translation effort. Following is an example of a MATLAB m-file that implements and verifies the above MathCAD $\Phi \mathbf{P} \Phi^T$ result.

Figure 2. *MATLAB Script Efficiently Unit Tests Ada Element*


MATLAB Script
Generate or Input Test Data
Write Test Data to File for Ada Test Driver
Execute FDD Algorithm in MATLAB Generating Expected Results
Pass Control to the Ada Unit Test Driver
Read Ada Test Results from File
Compare MATLAB Expected Results to Ada Unit Results
Save Test Results to File and Report Pass/Fail to Console
Ada
Input MATLAB Test Data from File
Execute Ada Unit Under Test
Write Ada Test Results to File for MATLAB Script
Return Control to MATLAB Script

$$\Phi P \Phi^T = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p11 & p12 & p13 & p14 & p15 & p16 \\ p12 & p22 & p23 & p24 & p25 & p26 \\ p13 & p23 & p33 & p34 & p35 & p36 \\ p14 & p24 & p34 & p44 & p45 & p46 \\ p15 & p25 & p35 & p45 & p55 & p56 \\ p16 & p26 & p36 & p46 & p56 & p66 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T =$$

MathCAD's Symbolic Evaluation

$$\begin{bmatrix} p11 + 2 \cdot dt \cdot p14 + dt^2 \cdot p44 & p12 + dt \cdot p24 + dt \cdot p15 + dt^2 \cdot p45 & p13 + dt \cdot p34 + dt \cdot p16 + dt^2 \cdot p46 & p14 + dt \cdot p44 & p15 + dt \cdot p45 & p16 + dt \cdot p46 \\ p12 + dt \cdot p24 + dt \cdot p15 + dt^2 \cdot p45 & p22 + 2 \cdot dt \cdot p25 + dt^2 \cdot p55 & p23 + dt \cdot p35 + dt \cdot p26 + dt^2 \cdot p56 & p24 + dt \cdot p45 & p25 + dt \cdot p55 & p26 + dt \cdot p56 \\ p13 + dt \cdot p34 + dt \cdot p16 + dt^2 \cdot p46 & p23 + dt \cdot p35 + dt \cdot p26 + dt^2 \cdot p56 & p33 + 2 \cdot dt \cdot p36 + dt^2 \cdot p66 & p34 + dt \cdot p46 & p35 + dt \cdot p56 & p36 + dt \cdot p66 \\ p14 + dt \cdot p44 & p24 + dt \cdot p45 & p34 + dt \cdot p46 & p44 & p45 & p46 \\ p15 + dt \cdot p45 & p25 + dt \cdot p55 & p35 + dt \cdot p56 & p45 & p55 & p56 \\ p16 + dt \cdot p46 & p26 + dt \cdot p56 & p36 + dt \cdot p66 & p46 & p56 & p66 \end{bmatrix}$$

```
% Define the number of test iterations and
% an acceptable tolerance for this test
Num_Iterations = 1000 ;
Tolerance      = 1e-014 ;

% Define 3x3 matrices and the sampling interval
I3 = eye(3) ;    % identity
Z3 = zeros(3) ;  % zeros
dt = 1/30 ;      % 30Hz

% Reset generator to initial state for repeatability
rand('state', 0) ;

% Define Phi, the 6x6 state transition matrix
Phi = [I3  I3*dt
       Z3  I3   ] ;

for Iteration = 1:Num_Iterations

  % Define a new random symmetrical 6x6 covariance matrix
  P = rand(6) ;                   % random 6x6
  U = triu(P,1) ;                 % upper triangular above
                                    main diagonal
  P = U + U' + diag(diag(P)) ;    % symmetric 6x6

  % Execute the simple form
  flops(0) ;             % reset flops counter
  P_Simple = Phi*P*Phi' ;
  Simple_flops = flops ; % accumulate flops

  % Execute the expanded form
  flops(0) ; % reset flops counter

  dt2 = dt*dt ;
  P_Expanded(1,1) = P(1,1) + 2.0*P(1,4)*dt + P(4,4)*dt2 ;
  P_Expanded(1,2) = P(1,2) + ( P(1,5) + P(2,4) )*dt +
                    P(4,5)*dt2;
  P_Expanded(1,3) = P(1,3) + ( P(1,6) + P(3,4) )*dt +
                    P(4,6)*dt2;
  P_Expanded(1,4) = P(1,4) + P(4,4)*dt ;
  P_Expanded(1,5) = P(1,5) + P(4,5)*dt ;
  P_Expanded(1,6) = P(1,6) + P(4,6)*dt ;

  P_Expanded(2,1) = P_Expanded(1,2) ;
  P_Expanded(2,2) = P(2,2) + 2.0*P(2,5)*dt + P(5,5)*dt2 ;
  P_Expanded(2,3) = P(2,3) + ( P(2,6) + P(3,5) )*dt +
                    P(5,6)*dt2;
  P_Expanded(2,4) = P(2,4) + P(4,5)*dt ;
  P_Expanded(2,5) = P(2,5) + P(5,5)*dt ;
  P_Expanded(2,6) = P(2,6) + P(5,6)*dt ;

  P_Expanded(3,1) = P_Expanded(1,3) ;
  P_Expanded(3,2) = P_Expanded(2,3) ;
  P_Expanded(3,3) = P(3,3) + 2.0*P(3,6)*dt + P(6,6)*dt2 ;
  P_Expanded(3,4) = P(3,4) + P(4,6)*dt ;
  P_Expanded(3,5) = P(3,5) + P(5,6)*dt ;
  P_Expanded(3,6) = P(3,6) + P(6,6)*dt ;

  P_Expanded(4,1) = P_Expanded(1,4) ;
  P_Expanded(4,2) = P_Expanded(2,4) ;
  P_Expanded(4,3) = P_Expanded(3,4) ;
  P_Expanded(4,4) = P(4,4) ;
  P_Expanded(4,5) = P(4,5) ;
  P_Expanded(4,6) = P(4,6) ;

  P_Expanded(5,1) = P_Expanded(1,5) ;
  P_Expanded(5,2) = P_Expanded(2,5) ;
  P_Expanded(5,3) = P_Expanded(3,5) ;
  P_Expanded(5,4) = P_Expanded(4,5) ;
  P_Expanded(5,5) = P(5,5) ;
  P_Expanded(5,6) = P(5,6) ;

  P_Expanded(6,1) = P_Expanded(1,6) ;
  P_Expanded(6,2) = P_Expanded(2,6) ;
  P_Expanded(6,3) = P_Expanded(3,6) ;
  P_Expanded(6,4) = P_Expanded(4,6) ;
  P_Expanded(6,5) = P_Expanded(5,6) ;
  P_Expanded(6,6) = P(6,6) ;

  Expanded_flops = flops ; % accumulate flops

  % Compare approaches
  Diff = P_Simple - P_Expanded ;

  % Ensure MATLAB and Ada implementations are equivalent
  if find(abs(Diff) > Tolerance)
    disp('Expanded implementation is not correct.')
    Diff     % display differences
    keyboard % debug mode on error
  end

  % Ensure expanded implementation enforces symmetry of P
  if find(abs(P_Expanded - P_Expanded') > Tolerance)
   disp('The expanded implementation violates symmetry of P')
    P_Expanded % display the covariance matrix
    keyboard   % debug mode on error
  end

  % Report flop results to console
  disp(['Iteration #', num2str(Iteration)])
  disp(['  Simple flops   = ' num2str(Simple_flops)])
  disp(['  Expanded flops = ' num2str(Expanded_flops)])
  disp(' ')

end % loop
```

MATLAB code to be translated into Target Language

In this sparse matrix example, for each iteration, MATLAB reports 864 FLOPs before the expansion and 49 FLOPs after, for a 17.6 : 1 savings in FLOPs!

Although the above example is quite simple and is almost as easily developed using pencil and paper, one quickly becomes severely entangled in tedious algebra when attempting, for example, to manually derive the Kalman gain matrix from $K = PH^T[HPH^T + R]^{-1}$ for even six states. (Here H is 3×n, P is n×n, R is 3×3 and K is n×3, where *n* is the number of states.) It is for examples similar to this that the power and value of the process are quickly substantiated. This is true even when the capabilities of MathCAD to display symbolic results are exceeded. In such cases, one simply subdivides the process into manageable portions. In the above Kalman gain equation, for example, intermediate expressions for PH^T are computed first, followed by $[HPH^T + R]$. Then $[HPH^T + R]^{-1}$ is derived and that result premultiplied by the PH^T term that was already expanded. Although this requires somewhat more work due to the maintaining of intermediate results, overall savings and code generation metrics are still very impressive. Depending on the particular implementation and the correlation of the measurements, FLOP savings on the order of 4:1 to 6:1 and more have been tabulated. These savings were achieved with significantly less effort and higher reliability than the manually tedious and error-prone traditional approach.

A process similar to the above is then used to unit test the algorithm in the target language. A test driver is created that executes the unit under test. Its executable image is called in place of the expanded code bracketed in the example m-file above. The MATLAB script and the target language test driver must both provide for the reading and writing of data files to effect the transfers of test data and results. The author has elected to mechanize these transfers in IEEE 64-bit binary format to take full advantage of the numeric capabilities of MATLAB. An outline of the MATLAB script was shown in Figure 2.

Although applications vary enormously in complexity, the author has experienced estimated savings from 40 to almost 80 percent in development and testing time (including debugging effort). For example, it took approximately 20 hours to manually develop and test code to decrement a 6-state covariance matrix and about 16 hours for the 6-state Kalman gain matrix. Applying the techniques outlined in this paper reduced these efforts to approximately 12 and eight hours, respectively. Without manually deriving corresponding 9-state equations, a fairly daunting task, extrapolations from actual 6-state manual results and other similar experience were used to estimate the level of effort that would be required for the manual derivation of 9-state equations. The following table summarizes and compares both approaches.

	Pencil & Paper	MathCAD & MATLAB	Estimated Savings
Decrement Covariance			
$[I-KH]P[I-KH]^T + KRK^T$			
6-State	20	12	40.0%
9-State	56*	16	71.4%
Compute Kalman Gain			
$PH^T[HPH^T + R]^{-1}$			
6-State	16	8	50.0%
9-State	42*	10	76.2%

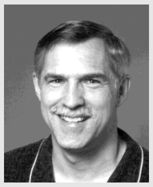
* Extrapolated from similar 6-State experience

Table 1. *Estimated Effort to Code and Test (hours)*

Summary

The use of commercial tools, such as MATLAB and MathCAD, can dramatically improve the efficiency of the software development process as well as the reliability of the final embedded product. Since systems engineering groups are increasingly using these tools to develop and document their products, it is becoming imperative that software engineers acquire the necessary proficiency to use and integrate these products into their processes. This is especially critical as contractors strive to remain competitive while meeting exigent schedules and maintaining budgetary constraints. Although not a panacea for all that is ailing in the software development process, the logical application of available commercial tools can have a significant, positive impact on that effort. This paper outlines such a process that the author has successfully applied on a current development program.

About the Author



Keith R. Wegner is a Fellow Engineer in the software engineering group at Northrup Grumman Corporation's Electronic Sensors and Systems Sector in Baltimore. He has a master's degree in electrical engineering from the Johns Hopkins University with emphasis in signal processing and control systems. He has used MATLAB for approximately 14 years.

Northrup Grumman Corp.
P.O. Box 746, MS-429
Baltimore, Md. 21203
Voice: 410-765-4664
Fax: 410-765-1492
E-mail: keith_r_wegner@mail.northgrum.com

Statistical Process Control Meets Earned Value, by Lipke/Vaughn, continued from page 20

References

1. Florac, William A., and Anita D. Carleton, *Measuring the Software Process*, Addison Wesley, Reading, Mass., 1999.
2. Pitt, Hy, *SPC for the Rest of Us*, Addison-Wesley, Reading, Mass., 1995.
3. Software Engineering Institute Course, Statistical Process Control for Software, July 1999.
4. Software Productivity Consortium Course, Statistical Process Control and Quality Management Techniques, August 1998.
5. Radice, Ron, Statistical Process Control for Software Projects, 3rd Annual Software Metrics Conference, December 1997.
6. Fleming, Quentin W., *Cost/Schedule Control Systems Criteria, The Management Guide to C/SCSC,* Probus, Chicago, 1988.
7. Lipke, Walter H., Applying Management Reserve to Software Project Management, CROSSTALK, March 1999.
8. Crow, Edwin L., Davis, Francis A., Maxfield, Margaret W., *Statistics Manual*, Dover Publications, New York, 1960.

Notes

1. To remove any confusion, by monthly performance values, we mean the values include only the performance occurring during the month.
2. The application of Table 1 in [7] required CPI^{-1} and SPI^{-1} to be cumulative values. For this application, CPI^{-1} and SPI^{-1} are average values of the monthly data.
3. The assumption in overtime and staffing equations is that the plan is being executed; i.e., the overtime rate and the staffing employed is in agreement with the project plan. If the effective values for either differ from the plan, it is recommended to use those values in the equations.
4. See the Table 1 management action description for the condition: cost comparison green, schedule comparison red.

About the Authors

Walt Lipke is the deputy chief of the Software Division at the Oklahoma City Air Logistics Center, which employs approximately 600 people, most of whom are electronics engineers. He has 30 years of experience in the development, maintenance, and management of software for automated testing of avionics. In 1993, with his guidance, the Test Program Set and Industrial Automation (TPS and IA) functions of the division became the first Air Force activity to achieve Software Engineering Institute Capability Maturity Model (SEI CMM®) Level 2. In 1996, these functions became the first software activity in federal service to achieve SEI CMM® Level 4 distinction. The TPS and IA functions, under his direction, became ISO 9001/TickIT registered in 1998. These functions were honored with the 1999 IEEE Computer Society Award for Software Process Achievement. Lipke is a professional engineer with a master's degree in physics.

Jeff Vaughn is the Metrics and Financial Analyst of the Oklahoma City Air Logistics Center's Software Division. He has 13 years experience in Avionics Test Program Set Development and Maintenance. He managed one of the first organizations in the Software Division to utilize EV Management techniques to manage projects. He has a bachelor's degree in electrical engineering.

OC-ALC/LAS
Tinker AFB, Okla. 73145-9144
Voice: 405-736-3335
Fax: 405-736-3345
E-mail: wlipke@lasmx.tinker.af.mil
jvaughn@lasmx.tinker.af.mil

Letter to the Editor

Dear CrossTalk:

Ever since I've been a CrossTalk reader, which has been around four to five years, I look forward to the next issue. CrossTalk is one of the best, if not the best, publications in the software process improvement (SPI) arena. There are always practical lessons learned in the wide variety of articles, something I have been able to use in my work. In addition, CrossTalk now includes Web links to many valuable sources of information.

As a former government employee and government contractor, I have been a frequent beneficiary of sharing information. Just because something was developed by the government does not mean it can not be used in private industry and vice versa. There is no sense in making the same mistakes. CrossTalk is a great help in sharing lessons learned in areas of value to our company—risk management, managing change, project management, and others related to SPI. Also, the CrossTalk staff has always been a pleasure to work with and has a positive, can do attitude. Thanks for helping those of us in the SPI trenches.

Darrell Corbin
The Boeing Company

Large Software Systems–Back to Basics

by John R. Evans
SPAWAR Systems Center

Our computer hardware is growing in power exponentially. We naturally expect to use this power on larger, more complicated problems. There is a problem, however. Software development methods that worked fine on small problems seem to not scale well.

The Problem and Partial Solution

Today when we launch a software project, its likelihood of success is inversely proportional to its size. The Standish Group reports that the probability of a successful software project is zero for projects costing $10 million or more [1]. This is because the complexity of the problem exceeds one person's ability to comprehend it. According to The Standish Group, small projects succeed because they reduce confusion, complexity, and cost. The solution to the problem of building large systems is to employ those same techniques that help small projects succeed—minimize complexity and emphasize clarity.

The goals, constraints, and operating environment of a large software system, along with its high-level functional specification, describe the requirements of the systems. Assuming we have good requirements, we can decompose our system into smaller subsystems. Decomposition proceeds until we have discrete, coherent modules. The modules should be understandable apart from the system and represent a single idea or concept. When decomposition is finished, the modules can be incorporated into an architecture.

Frederick P. Brooks said that the conceptual integrity of the architecture is the most important factor in obtaining a robust system [2]. Brooks observed that it can only be achieved by one mind, or a very small number of resonant minds. He also made the point that architectural design must be separated from development. In his view, a competent software architect is a prerequisite to building a robust system.

An architecture is basically the framework of the system, detailing interconnections, expected behaviors, and overall control mechanisms. If done right, it lets the developers concentrate on specific module implementations by freeing them of the need to design and implement these interconnections, data flow routines, access synchronization mechanisms, and other system functions. Developers typically expend a considerable amount of energy on these tasks, so not doing them is a considerable savings of time and effort [3].

A robust architecture is one that is flexible, changeable, simple yet elegant. If done right and documented well, it reduces the need for interteam communication and facilitates successful implementation of complex modules. If done well, it is practically invisible; if done poorly, it is a never-ending source of aggravation, cost, and needless complexity.

Architecture flows from the requirements and the functional specification. The requirements and functional specification need to be traced to the architecture and its modules, and the modules in the architecture should be traced to the requirements and functional specification. The requirements must necessarily be correct, complete, unambiguous, and, where applicable, measurable. Obtaining requirements with these qualities is the responsibility of the architect. It must be his highest priority. He does this by interacting closely with the customers and domain experts. If necessary, he builds prototypes to validate and clarify the requirements The architect acts as the translator between the customers and the developers. The customers do not know how to specify their needs in the unambiguous language that developers need, and the developers do not always have the skills to do requirements analysis.

The architect communicates his desires to the developers by specifying black-box descriptions of the modules. Black boxes are abstract entities that can be understood, and analyzed independently of the rest of the system. The process of building black-box models is called abstraction. Abstraction is used to simplify the design of a complex system by reducing the number of details that must be considered at the same time, thus reducing confusion and aiding clarity of understanding [4]. For safety-critical, military-critical, and other high-integrity sytems, black boxes can be specified unambiguously with mathematical logic using formal methods. Supplemented with natural language descriptions, this is probably the safest way to specify a system. It is usually more expensive and time consuming, as well. In the future, however, all software architects should know how to mathematically specify a module.

A robust architecture is necessary for a high-quality, dependable system. But it is not sufficient. A lot depends on how the developers implement modules handed to them by the architect.

The Rest of the Solution

Developers need to build systems that are dependable and free from faults. Since they are human, this is impossible. Instead they must strive to build systems that minimize faults by using best practices, and they must use modern tools that find faults during unit test and maintenance. They should also be familiar with the concepts of measuring reliability and how to build a dependable system. (A dependable system is one that is available, reliable, safe, confidential, has high integrity, and is maintainable [5].) In order for the system to be dependable, the subsystems and modules must be dependable.

"A key point [Hatton] stresses is to never incorporate stylistic information into the standard."

Fault prevention starts with clear, unambiguous requirements. The architect should provide these so the developer can concentrate on implementation. If the architecture is robust, the developer can concentrate on his particular module, free of extraneous details and concerns. The architect's module description tells the developer *what* to implement, but not *how* to implement it. The internals of the implementation are up to him. To ensure dependability, the developer needs to use sound software engineering principles and best practices, as these are his chief means of of minimizing complexity. Two best practices are coding standards and formal inspections.

Coding standards are necessary because every language has problem areas related to reliability and understandability. The best way to avoid the problem areas is to ban them, using an enforceable standard. Les Hatton describes why coding standards are important for safety and reliability and how to introduce a coding standard [6]. A key point he stresses is to *never* incorporate stylistic information into the standard. It will be a never-ending source of acrimony and debate. Such information, he says, should be placed in a *style guide.* Coding standards can be enforced with automatic tools that check the code, and by formal inspections. The benefits of formal inspections for defect prevention are well-known and well-documented. They are also invaluable for clarifying issues related to the software.

Developers need to measure their code to ensure its quality. This provides useful feedback to the developer on his coding practices, and it provides reassurance to the system's acquirers and users. Many static metrics can be used to assess the code. Among these are purity ratio, volume, functional density, and cyclomatic complexity. As a doctor uses a battery of tests to gauge a person's health, relying on more than one metric and covering all his bases, a developer using static analysis tools can do the same [7].

A good metric, for example, is cyclomatic complexity. A large value is a sign of complex code, which may be an indication of poor thought given to the design and implementation. It is also a sign that the code will be difficult to test and maintain.

Fault detection by proper unit testing is vitally important. To be done right, it requires the use of code coverage and path analysis tools. Unfortunately, this type of testing is usually overlooked. Many managers say they cannot afford them. Somehow, though, they can afford to fix the problems after the software has been fielded. This is penny-wise and pound-foolish. It is axiomatic that fixing software faults after the code has been deployed can be up to 100 times more expensive than finding and fixing the fault during development [8].

Besides path analysis and code coverage tools, automatic testing tools should be used. Human testers cannot hope to match the computer on indefatigability or thoroughness. In large systems, if testing is not automated, it is not done, or done rarely. For example, regression testing, used in systems undergoing modification and evolution, is essential to ensure that errors are not injected into code undergoing change, a very common problem in complex systems. Without automation, the process is onerous and time consuming. It rarely gets done, if at all.

Developing quality code is not simple or easy. It requires discipline and rigor, state-of-the-art tools, and enlightened managers willing to support developers by paying up-front costs, such as giving developers more time to automate and test their code. Developers take pride in their work. When they get the support they need, they know that their managers want them to produce quality code. This makes the work satisfying and rewarding.

Summary

Managing and limiting complexity and promoting clarity is fundamental to developing large software systems. The key ingredient is a robust architecture. The conceptual integrity of the architecture, its elegance and clarity, depends on a single mind. Developers build upon the architecture and ensure its robustness by rigorous application of basic software engineering principles and best practices in their code development.

References

1. Johnson, J., *Turning Chaos into Success,* www.softwaremag.com/archive/1999dec/Success.html, Dec. 1999. Standish Group.
2. Brooks, F. P., *The Mythical Man-Month: Essays on Software Engineering,* Anniversary Edition. Addison-Wesley, 1995.
3. Bass, L., P. Clements, and R. Kazman, Software Architecture in Practice. Addison-Wesley, 1998.
4. Berzis, V., and Luqi, *Software Engineering with Abstractions.* Addison Wesley, 1991.
5. Lyu, Michael R., Editor, *Handbook of Software Reliability Engineering.* IEEE Computer Society Press, 1995.
6. Hatton, L., Safer C, *Developing Software for High-Integrity and Safety-Critical Systems.* McGraw-Hill International Series in Software Engineering, McGraw-Hill International, 1995.
7. Drake, T. Measuring Software Quality: A Case Study, *IEEE Computer,* Nov. 1996.
8. Boehm, B. W., *Software Engineering Economics.* Prentice Hall, 1981.

About the Author

John Evans is a software engineer at SPAWAR Systems Center in San Diego (SSC-SD), where he has worked for the last 16 years. His job is to improve the software processes of projects within the Intelligence, Surveillance, and Reconnaissance Department (D70), and help the Software Engineering Process Office (SEPO) of SSC-SD improve the center's software maturity. He received his master's degree in software engineering from the Naval Postgraduate School in 1997 via distance learning, and SSC-SD sponsorship. He is now working on his doctorate in software engineering, under the same auspices.

SPAWARSYSCEN D73C
53570 Silvergate Ave., Room 1047
San Diego, Calif. 92152-5182
Voice: 619-553-5479
Fax: 619-553-5499
E-mail: evansjr@spawar.navy.mil

Give Us Your Information, Get a Subscription

Fill out and send us this form for a free subscription.

OO-ALC/TISE
7278 FOURTH STREET
HILL AFB, UTAH 84056-5205
ATTN: HEATHER WINWARD
FAX: 801-777-8069 DSN: 777-8069
VOICE: 801-775-5555 DSN: 775-5555

Or use our online request form at www.stsc.hill.af.mil

FULL NAME:______________________________

RANK OR GRADE:__________________________

POSITION OR TITLE:_______________________

ORGANIZATION OR COMPANY:_______________

ADDRESS:_________________________________

____

BASE OR CITY:____________ STATE:_____

ZIP:__________

VOICE: COMMERCIAL___________________

DSN___________________

FAX: COMMERCIAL___________________

DSN___________________

E-MAIL: ________________@________________

BACK ISSUES MAY BE AVAILABLE

(PLEASE INDICATE THE MONTH(S) DESIRED.)

____

____

____

____

Common Sense–Can You Dig It?

Processes are a good thing. I am a PSP instructor, and appreciate how a good process can make my work habits more productive and increase personal quality. In fact, I was discussing with my lovely and charming wife about my idea to create a Personal Garden Planting Process (PGPP). She was quite amused with my PGPP, but pointed out that the process was quite useless.

"Never," I countered to this heresy. I had considered everything from soil moisture to length of the grass. She pointed out that there were several inches of snow in the yard—my process had neglected to consider the uselessness of planting a garden in the middle of winter. Good process—poor timing and implementation. Which brings us to Mr. Adams, Mr. Baker, and Mr. Charles

Mr. Adams, Mr. Baker and Mr. Charles were three software engineers in a particular Department of Defense agency who suddenly found themselves without a job. It seems the agency, after doing an A-76 study, contracted out the work and transferred all but these three software developers to other jobs.

Unfortunately, these men were within three months of retirement. Management, the big softies, arranged to have them perform other chores within the organization for three months, and then take retirement. Mr. Adams, Mr. Baker, and Mr. Charles were offered the job of site beautification. Their job was to plant shrubbery around the various buildings.

Being trained software engineers, the three men got together and came up with a process. Every evening, they ran the automatic sprinkler system to soften the ground. Early in the morning, Mr. Adams would start digging the holes for the shrubs. Mr. Baker would follow him, spreading fertilizer and inserting the shrubbery. Finally, Mr. Charles would cover the roots with dirt.

One day while parking his car, the big boss was puzzled to see Mr. Adams busy digging holes that Mr. Charles immediately filled with dirt. He expressed his amazement, only to be told that Mr. Baker was ill that day. But as Mr. Adams and Mr. Charles explained, "There's no reason to abandon our process just because one person isn't following it."

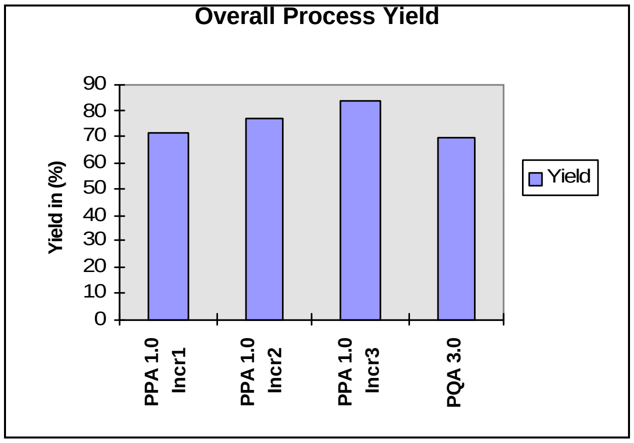
It is important to have a process—it serves as a road map. It lets you know how to get to where you are going. If you don't know where you are going, no matter how good the road map is, it doesn't help. It is important to understand the purpose of the road map, your goals, and the rationale of your process. Know when to follow the process—and know when the process will not work. When it does not work, it is time to modify the process. Processes are important. So is common sense.

—Dave Cook, Draper Laboratories Inc.

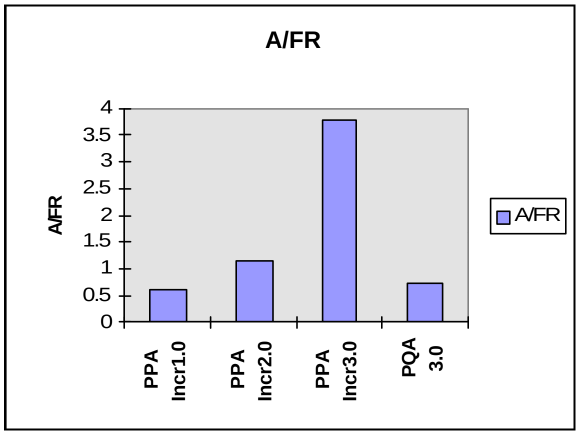
CALL FOR ARTICLES

software acquisition	Submit by July 1, 2000
project management	Submit by Aug. 1, 2000
modeling & simulation	Submit by Sep. 1, 2000
configuration management	Submit by Oct. 1, 2000
measures & metrics	Submit by Nov. 1, 2000
requirements management	Submit by Dec. 4, 2000

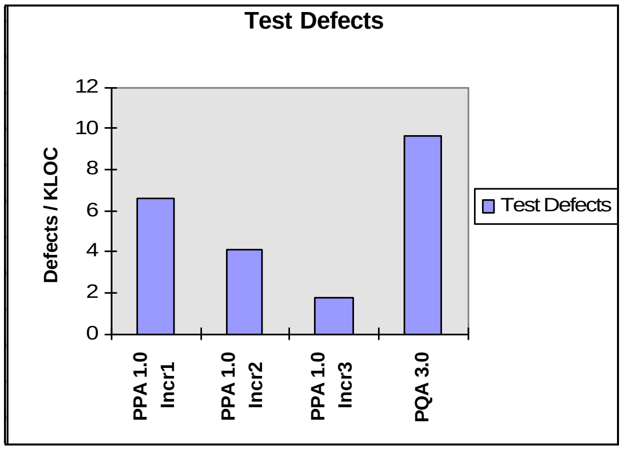
E-mail submissions to features@stsc1.hill.af.mil
Author Guidelines may be found at www.stsc.hill.af.mil



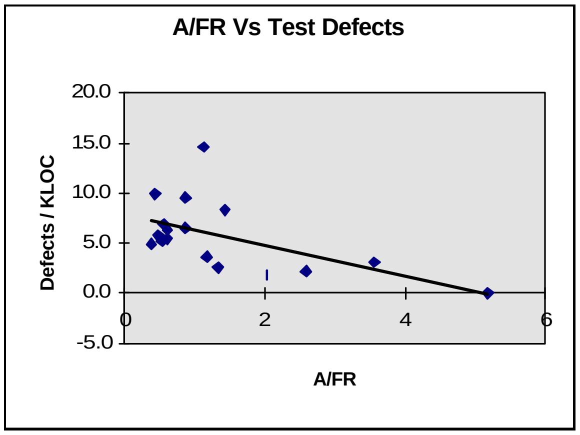
Higher Yields



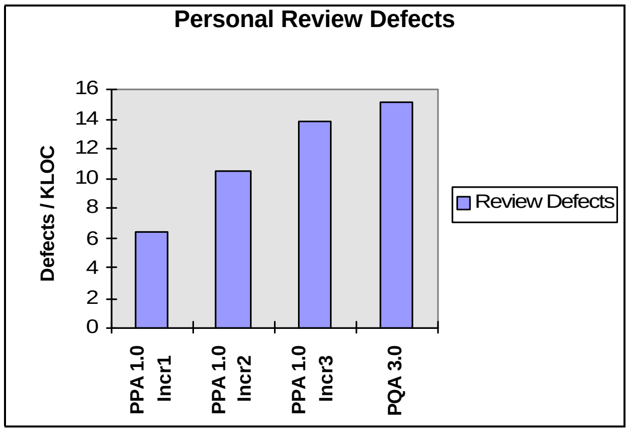
Greater A/FR



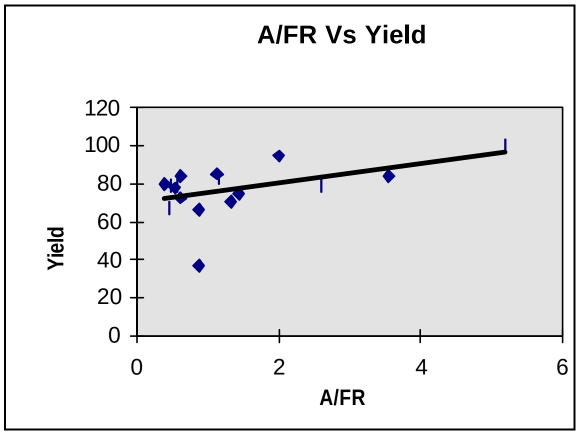
Fewer Test Defects



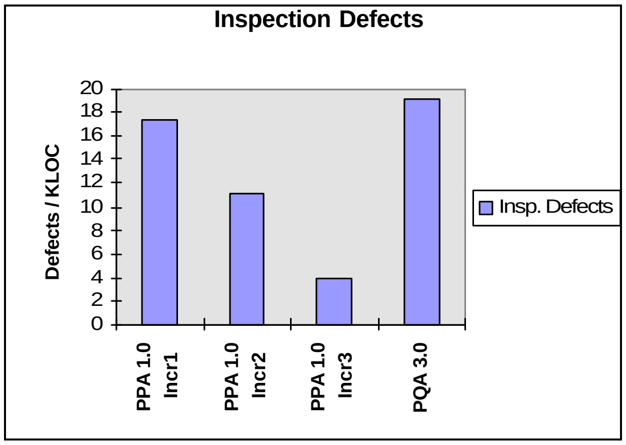
A/FR Vs. Test Defects



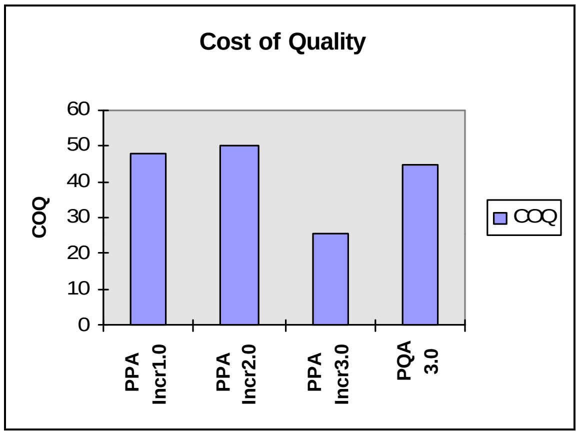
More Defects found in Personal Reviews



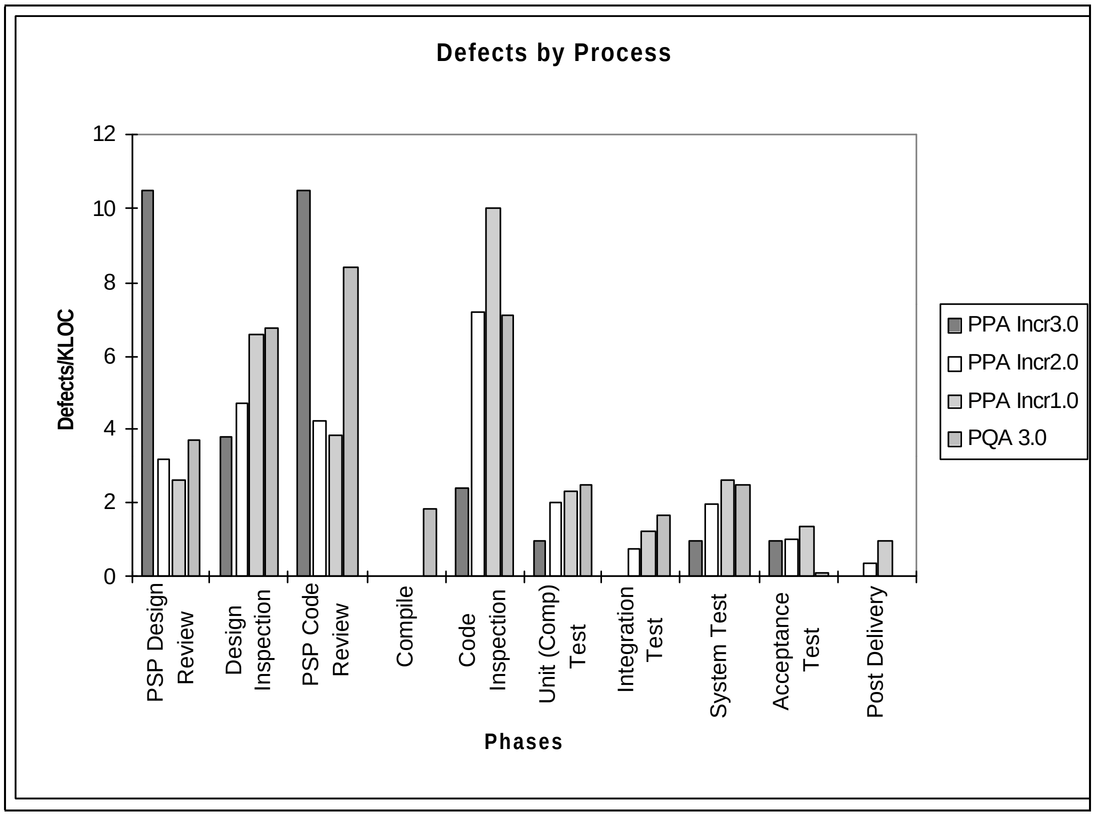
A/FR Vs. Yield



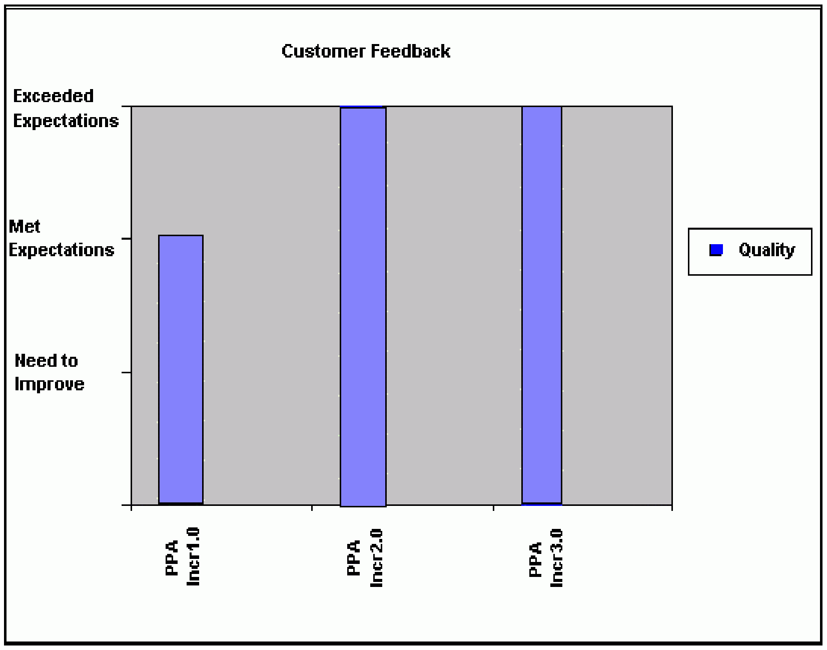
Fewer Defects found in Team Inspections



Cost of Quality Declined



Defects by Process



Customer Expectations Exceeded

The V Model

by Morton Hirschberg
Formerly of the Army Research Laboratory

The author is the technical project officer for the Data Exchange Agreement for Software Technology between the United States and Germany. It was in this capacity that he became aware of the German software standards, known as the V Model, for the German Federal Armed Forces. The standards are published in three volumes and can be tailored to fit officially sponsored work. In this Web Addition, the author introduces these standards to give readers a flavor for them and to encourage learning more about software standards used by a political and military ally.

It is not the author's intention to contrast the V Model with U.S. standards and directives, nor to comment about their use by the German Federal Armed Forces. The characterizations, such as strengths, are loosely quoted from summaries written by Herr Fritz Haertel, one of the architects of the V Model. They do not necessarily reflect the author's viewpoint.

The V Model

The V Model is a series of General Directives (250, 251, and 252) that prescribe or describe the procedures, methods to be applied, and the functional requirements for tools to be used in developing software systems for the German Federal Armed Forces.

General Directive 250. August 1992. Software Lifecycle Process Model.

The objective of this directive is to regulate the software development process by a uniform and binding set of activities and products that are required during software development and its accompanying activities. Use of the V Model helps to achieve:

1) Improvement and warranty of software.
2) Reduction of software costs for the entire life cycle.
3) Improvement of communications among the different parties as well as the reduction of the dependence of the customer upon the contractor.

The V Model deals with procedure, method, and tool requirements. Its main advantage is that it can be generally applied. It fits into the international picture by fulfilling requirements of NATO standards, ISO 9000 technical standards, and the structure of the EURO-METHOD. None of these is discussed here, but they could be featured in subsequent articles.

The V Model organizes all activities and products and describes activities and products at different levels of detail. In the V Model, products take on one of four states:

1. Planned: the initial state of all products
2. Processed: either in private development or under the control of the developer
3) Submitted: completed and now ready for quality assessment. It can be returnedto the processing stage if rejected or advance to accepted for release.
4) Accepted.

While seemingly prescriptive, the V Model allows for tailoring throughout the product life cycle. That is one of its strengths. The V Model is composed of four submodels: software development, quality assurance, configuration management, and project management.

The submodels are closely interconnected and mutually influence one another by exchange of products and results. The software development submodel develops the system or software. The quality assurance submodel submits requirements to the other submodels and test cases and criteria to assure the products and the compliance of standards. The configuration management submodel administers the generated products. The project management model plans, monitors, and informs the other submodels. Each can be further decomposed. For instance, the software development submodel can be broken down as follows:

- System requirements analysis and design.
- Data processing requirements analysis and design.
- Software requirements analysis.
- Preliminary design.
- Detailed design.
- Implementation.
- Software integration.
- Data processing integration.
- System integration.

The directive contains very detailed information (rules) on the activities of each submodel showing product flow, handling, and, if warranted, recommendations. For example, in the planning stage the product flow is from an external specification to a submitted state. Handling consists of organization planning, cost and scheduling, resource planning, and risk considerations.

There may also be peculiarities that need to be addressed. Recommendations that might be considered are:

- Use of matured resources and an experienced staff.
- Correct membership participation in cost and planning
- Scheduling.
- Consideration of alternative problem solutions.
- Knowing how to handle unexpected events
- Considering costs for management activities and coordination activities.

Occasionally, the directive also includes further explanations.

It should be noted that prototyping can be used to verify and detail requirements. Prototyping allows for early completion, as an aid in refining requirements, feasibility, and testing.

Each directive has a set of appendices containing definitions, a list of abbreviations, a list of illustrations, a bibliography, a characterization of the roles of the products, a list of activities to be performed, a list of products, an index and annexes. Directive 250 has two annexes.

The purpose of Annex 1 is to provide explanations of the application of the V Model. It is to support the user and is not of a binding nature. The objective of the V Model is to submit the products created during software development, maintenance, and modification to certain standards. This is to guarantee a minimum quality of the results and to make it easier to control the product stages from requirements definition to the-final product itself. The V Model offers support as a guide, as a checklist, and for the quality goal definition. The V Model allows for tailoring, defines required products, and establishes criteria for assessment.

Two kinds of applications have been intended for the V Model—as a basis for contracts and as a development guide. The V Model makes provisions for the use of commercial, non-developed items, and commercial off-the-shelf software. It also provides for information technology projects.

Annex 1 also provides the elements that may be deleted from the model.

Annex 2 is an explanation of the products. It deals with reports and software to be produced. This includes requirements, architectures, and design. It covers user, diagnostic, and operator manuals. It also is broken down by the same four submodels.

General Directive 251. September 1993. Methods Standard.

The objective of this standard is to set down all the tasks and results ofthe software development process. Standardization is done on three levels: procedure, methods to be applied, and functional requirements on tools to be used. While the V Model answers what is to be done, the Methods Standard answers *how* it is to be done.

More than 30 basic methods or categories of methods are listed in the standard. These are, for example, bar charts, tree diagram, decision table techniques, E(ntity)/R(elationship) modeling, normalization, object design technique, simulation models, and structured design.

The Methods Standard includes allocation tables listing those basic methods that are best suited to realize certain activities or to develop products according to the latest state of the art and by observing the criteria of quality improvement, economy, and maintainability. For each method referenced in the allocation tables, the standard describes the features that an applied method must include to reach the standard. In many instances a complex method may be required. This may represent a well-defined combination of several basic methods. Basic methods really refer to procedures that describe a special, limited aspect of a system such as, functionality, data orientation, analysis, preliminary design, or one of the activities—quality assurance, configuration management, or program management. Complex methods usually cover several aspects of a system.

Basic methods must be applied unless, by limiting conditions, they make applying the method impractical, or there are arguments against the method or for an alternative method in a special case. Each method listed includes identification/definition, characteristics, limits, specification, interfaces, and a list of references. The Methods Standard is not meant to be a methods manual. Regarding tools, a method may be applied in different versions depending upon the manufacturers. For this reason, tool-independent definitions are set up.

Allocation tables exist for software development, quality assurance, configuration management, and project management.

The Methods Standard can be made modified by a Change Control Board that meets annually and is made up of industry and government. Besides the main part of the Method Standard, there are two annexes.

Annex 1 provides an explanation of the methods; the method interfaces including a characterization of the interface, an example of the interface, tool support, relevant literature, and a description of the methods. The methods explanation contains information about technical and operational software inspection and walkthroughs.

Annex 1 addresses object design technique and configuration management. It further contains a section on estimation models (function point method, constructive cost model), simulation models (continuous and discrete—time-controlled, event-driven, activity-oriented, process-oriented, or transaction-oriented), system behavior models (Petri networks, state charts, specification and description language),and reliability models (statistic [Okumoto, execution time, Logarithmic Poisson, Jelinski-Moranda, and Schick and Wolverton], and error seeding).

Annex 2 helps when applying complex methods in connection with the software development standard. This annex describes the most important methods for application in German national projects. The methods are:

1) Graphical Engineering System (GRAPES)
2) Information Engineering Method (IEM)
3) Integrated Software Technology (ISOTEC)
4) The quality management system of the CAP Gemini Group (PERFORM)
5) Structured Analysis and SA with Real Time Extensions (SA & SA/RT)
6) Specification and Design Language (SDL)
7) Software Engineering Technology (SEtec)
8) Structured Systems Analysis and Design Method(SSADM)

For each of the above, a brief description, tabular comparison with the basic methods, specification of the allocation, and relevant literature is given.

General Directive 252. September 1993. Functional Tool Requirements (Standardized Criteria Catalogue).

The goal of this standard is to constrain the variety of applied methods and tools that can be employed during the software life cycle. While theV Model answers what is to be done and the Methods Standard answers how it is to be done, the Functional Tool Requirements answers *with what* it is to be done.

The standard increases the guarantee for software quality (higher quality products, lower risk), minimizes software cost for the entire life cycle, imposes communication among the different parties, and reduces dependence of the customer on the contractor. The latter is accomplished through its recommendations, focused approach, and required prescriptions.

The standard introduces the software development environment (SDE) reference model where SDE is defined as "the totality of all technical resources utilizedfor the professional software development." A tool is defined as "a software product supporting the generation or maintenance or modification of software."

The structure of the SDE reference model is:

- User interface.
- Work flow management.
- Security and integrity requirements.
- Software development.
- Quality assurance.
- Configuration management.
- Project management.
- Object management.

The description of the fundamental units or criteria—or service units, as they are referred to in the standard—are laid out as allocation to the V Model and Methods Standards, brief characteristics, and finally, requirements.

The reference model puts all the technical data processing services offered into a basic schema. Fifty-eight service units are defined and explained. A service unit can cover exactly one method or several methods. It should be noted that a method can be covered by one or more service units or not covered at all. In addition, there may be other requirements that are not based on a method. Finally, the Methods Standard may not suggest a method for the item under consideration. Some examples of service units are:

- From the user interface—help functions.
- From software development—generating databases, compiling, and debugging.
- From quality assurance—static assessment of the code.
- From project management—cost estimation.

An example of a service unit schema for cost estimation is allocation—planning, detailed planning, and estimation models; brief characteristics—requirements on tools to support the cost estimation realized by basis of already available empirical values from earlier projects, project specific marginal conditions, and by assumptions of future developments. For requirements—granularity, input and output interfaces to other service units, estimation models for fixed and variable costs, and other requirements such as an experience database.

The standard has an appendix and two annexes. An important part of the appendix is the relationship between the V Model and the European Computer Manufacturers Association (ECMA) Reference Model. The services in the ECMA Reference Model are:

- Object management.
- User interface.
- Process management.
- Policy enforcement.
- Communication.

Tools per se are not further specified in the ECMA Reference Model. There is no strict one-to-one correspondence between the V Model and the ECMA Reference Model.

Finally, Annex 1 supports the user in his work with functional tool requirements by means of tabular overviews and applications scenarios. The latter covers the definition of the functional requirements on a tool, the selection of tools for setting up a SDE, tool evaluation, and the tool environment in a customer/contractor relationship.

Annex 2 is a used as an introduction into the basics of SDE data management and to offer an overview of standards for the integration of tools with regard to data, control information, and user interface. Data management is handled through the use of data models. The real world is first portrayed in a conceptual design from which a logical design of relevant features is developed. Annex 2 provides definitions of a data dictionary, repository, and development data base.

Finally, the appendix deals with standards. Not all requirements can be met by a single tool, so a SDE is only possible if tools can be integrated into a uniform environment. Such integration has three aspects: data integration, control integration, and uniform user interface. The concentration is on data integration.

Several standardization efforts in the DP industry are discussed, including those of the Object Management Group.

Model Summary

The V Model, Methods Standard, and Tool Standard present complete coverage of the functional areas sSoftware development, quality assurance, configuration management, and project management), provide concrete support, is sophisticated, yet flexible and balanced, has a wide spectrum, and is publically controlled under the supervision of a Change Control Board. Improvements as well as corrective changes are handled through the Control Board.

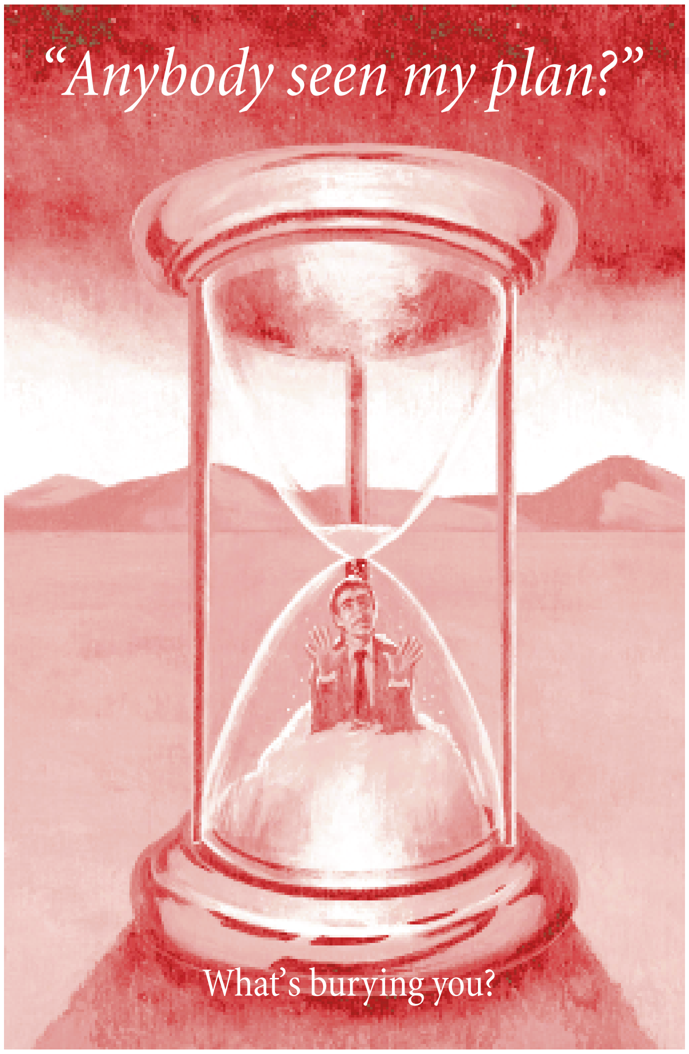
The advantages are improved communications among project members, uniform procedures, guarantee of a better product, productivity increases, better choice of methods, adaptability, reduced risk, lowered training costs, anddecreased maintenance.

Conclusion

It is my hope that the models presented can serve as a catalyst and framework for discussion of standards methodologies for the Department of Defense. It should be noted that the German Ministry of Defense, while similar to the Department of Defense, is much more homogeneous. Perhaps this is a major contribution to the use of their V Model.

Contact Information

Morton Hirschberg
207 Briarcliff Lane
Bel Air, Md. 21014-5524
E-mail: mortfran@aol.com



Helping Organizations Buy and Build Software Better

through

Process Improvement
Project Management
Systems Engineering
Software Acquisition
Capability Assessments

keeping you informed with

CrossTalk: The Journal of Defense Software Engineering

Software Technology Conference

Published by the Software Technology Support Center

S T S C

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

S T S C

www.stsc.hill.af.mil

Software Technology Support Center

OO-ALC/TISE
7278 4th Street
Hill Air Force Base, Utah 84056

801-775-5555
DSN: 775-5555
Fax: 801-777-8069

CrossTalk
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205

BULK RATE
US POSTAGE PAID
Permit No. 481
Cedarburg, WI